DOCUMENT RESUME

ED 281 505 IR 012 622

AUTHOR	Kingsbury, Douglas Lee
TITLE	Computer-Generated, Three-Dimensional Character Animation: A Report and Analysis.
PUB DATE	85
NOTE	93p.; Master's Thesis, Ohio State University. Illustrations may not reproduce.
PUB TYPE	Dissertations/Theses - Master Theses (042)
EDRS PRICE	MF01/PC04 Plus Postage.
DESCRIPTORS	*Animation; *Characterization; *Computer Graphics; *Production Techniques; *Three Dimensional Aids
IDENTIFIERS	Data Management; Ohio State Univ Computer Graphics Research Group

ABSTRACT

This master's thesis details the experience gathered in the production "Snoot and Muttly," a short character animation with 3-D computer generated images, and provides an analysis of the computer-generated 3-D character animation system capabilities. Descriptions are provided of the animation environment at the Ohio State University Computer Graphics Research Group, where the production was designed, and the making of "Snoot and Muttly," including the story, design of characters and environment, the storyboard, the animation process, image calculation and storage, and video production. Discussion of animation techniques and functions in the field of computer graphics includes analyses of motion control methods, character structure, and modeling schemes found in current animation systems. A bookkeeping system is proposed to manage the large amount of data involved in the production of character animation, and specifications for a number of animation system functions of particular value to the character animator are examined. Information on the practical aspects of character animation is presented for software developers, and animators are encouraged to participate in the animation system design process. The text is illustrated with 7 figures, 21 references are listed, and appendices contain sample storyboard sketches, business sketches, plans for data control points, and movement paths. (MES)

****
*   Reproductions supplied by EDRS are the best that can be made   *
*   from the original document.   *
****

ED281505

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION
CENTER (ERIC)

☒ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

- Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

COMPUTER-GENERATED, THREE-DIMENSIONAL CHARACTER ANIMATION: A REPORT AND ANALYSIS

A Thesis

Presented in Partial Fulfillment of the Requirements
for the Degree Master of Arts

by

Douglas Lee Kingsbury, B.A.

The Ohio State University
1985

Approved by
Thomas E. Linehan
Adviser
Department of Art Education

Copyright (c) 1985
by Douglas L. Kingsbury
All rights reserved.

"PERMISSION TO REPRODUCE THIS
MATERIAL HAS BEEN GRANTED BY
Douglas L.Kingsbury

TO THE EDUCATIONAL RESOURCES
INFORMATION CENTER (ERIC)."

IR012622

BEST COPY AVAILABLE

THESIS ABSTRACT

THE OHIO STATE UNIVERSITY

GRADUATE SCHOOL

NAME: Douglas Lee Kingsbury QUARTER/YEAR: Winter 1985

DEPARTMENT: Art Education DEGREE: Masters of Arts

TITLE OF THESIS:
Computer-Generated, Three-Dimensional Character Animation: A Report and Analysis

This thesis reports on the production of the computer-generated film, "Snoot & Muttly", and provides an analysis of computer-generated, three-dimensional character animation system capabilities. The animation environment at the The Ohio State University, Computer Graphics Research Group is reviewed. The thesis analyzes motion control methods, character structure, and modeling schemes found in current animation systems. A bookkeeping system for animation is proposed. The thesis concludes by examining animation system functions of particular value to the character animator. This thesis encourages animators to participate in the animation system design process and presents information for software developers regarding the practical aspects of character animation.

Thomas E. Linehan
Adviser's Signature

IR012622

Acknowlegments

My thanks to Susan Van Baerle, without whom "Snoot & Muttly", the initial impetus of this thesis, would not have been concieved. Thanks also to John A. Berton for the composition and performance of the original music for the animation. I would like to thank my advisor, Dr. Thomas Linehan, and the director of the Computer Graphics Research Group (CGRG), Professor Charles Csuri, for their advice and support. All of the people of CGRG and Cranston/Csuri Productions (CCP) have my deepest appreciation for their generousity with their knowledge and experience. I'd like to specifically thank Susan Amkraut, John Berton, Michael Girard, Julian Gomez, Brian Gunther, Paul MacDougal, Judy Sachter, Marla Schweppe, and David Zeltzer of CGRG, and Wayne Carlson, Scott Dyer, Shaun Ho, Tom Longtin, and Wayne Tyler at CCP. My thanks to CCP for the donation of computer time and equipment for final image calculation and recording of "Snoot & Muttly". I want to thank my friend, Jerry Holt, for his constant support. Finally, I'd like to thank my parents, Roger and Phyllis Kingsbury, for their continuing support of my endeavors.

Table of Contents

Acknowledgements ii
List of Figures v
Chapter 1. Introduction 1
Chapter 2. The Environment 4
Chapter 3. Making 'Snoot & Muttly' 7
3.1. The Story 7
3.2. Design of Characters and Environment 7
3.3. Animation System and Character Design Test 13
3.4. The Storyboard 14
3.5. The Animation Process 15
3.6. Image Calculation and Storage 22
3.7. Video Production 23
Chapter 4. Special Purpose Programs 25
4.1. Pathwalker 25
4.2. Bubble Flight 26
Chapter 5. Compromises 29
Chapter 6. The Character Animation System 33
Chapter 7. Control Methods 34
7.1. Interpolation 34
7.2. Movement Functions 36
7.3. Goal-Directed Control 37
7.4. Constraints 39
7.5. Simulation 40
Chapter 8. Character Structure 43
8.1. Rigid objects 43
8.2. Plastic objects 43
8.3. Rigid articulated figures 45
8.4. Plastic Articulated Figures 46
8.5. Composite Articulated Figures 47
Chapter 9. Modeling Schemes 48
9.1. Polygonal Mesh 48
9.2. Mathematically described Surfaces 48
9.3. Particle Systems 50
9.4. Density Functions 50
9.5. Volume Modeling 50
Chapter 10. Bookkeeping 51
10.1. Track Elements for Each Animation 51
10.2. System Organization 53
10.3. Common Data Area. 54
Chapter 11. System Specifications 55
11.1. Animate Characters Individually or in Combination 55
11.2. Solid Modeling Scheme Independence 56

11.3. Rendering Considerations 56
11.4. Multiple Control Methods 58
11.5. Planned System Growth 63
11.6. Object Substitution. 64
11.7. Interface 64
11.8. Human Readable/Editable Script 64
11.9. Device independence. 65
11.10. Frame Independence 66
11.11. Temporal Aliasing 67
11.12. Velocity control 68
11.13. Screen Space Composition Control. 68
11.14. Post-Processing Functions 69
11.15. Selective Image Generation 69
11.16. Low-Resolution, Real-Time, Raster Image Playback 69
11.17. The Cyclical Design Process 70
11.18. Future Challenges 72
11.19. Conclusion 72
List of References 74
BIBLIOGRAPHY 80
Appendix A: Storyboard Sketches 82
Appendix B: Business Sketches 83
Appendix C: Plans for Data Control Points 84
Appendix D: Sample Movement Paths 85

List of Figures

Figure 1. Snoot and Muttly 3
Figure 2. Legs, at Zero and in Walk Cycle 9
Figure 3. Legs in Attached Position and Construction of Snoot 11
Figure 4. Close-up of Snoot's Head 20
Figure 5. Snoot Constructed from Simplified Data 21
Figure 6. Muttly and Snoot with Bubbles 27
Figure 7. Elapsed Frame 30

CHAPTER 1

Introduction

This report details the experience gathered producing a short character animation, "Snoot & Muttly", with 3-d, computer generated images. The animation was a collaborative effort between Susan Van Baerle and the author. The music was composed and performed by John A. Berton, Jr. The music will not be discussed. Information on aspects of the production of "Snoot & Muttly" not treated in this thesis may be found in Van Baerle's thesis [56].

The animation continues the tradition of two-dimensional "cel" animators. The goal was to produce believable characters with personality. Snoot and Muttly had to be accepted as real. They had to have personalities that the audience could recognize and understand. They had to move in ways that would seem natural, given their unnatural physical construction. A great emphasis was placed on non-mechanical (non-linear, non-patterned) movement. Snoot and Muttly were to appear as self-directed, thinking creatures. They were to seem more organic than the rigid object construction typical of computer-generated characters.

The bulk of computer graphics animation possesses little "life". Computer-generated, raster graphics animation is mainly devoted to commercial advertising or research. Current commercial uses are commonly limited to animating a rigid logo. The remaining animation tends to be demonstrations of singular animation techniques.

Few entertaining animations are produced for the purpose of telling a story through the interaction of animated characters. Many of the animated characters that are produced are rotoscoped, or traced, from live-action footage of actors. Production of animation that is not rotoscoped requires the solution of problems within the tradition of character animation, as well as problems specific to computer graphics animation.

In the process of animating several characters to tell a story with computer graphics animation, we encountered many of these problems. Some of these problems we were able to solve, others we only identified. Some of these problems were specific to our work and resulted from our approach to computer-generated, character animation. But many were indigenous to the process of character animation.

This thesis outlines the process of animating "Snoot & Muttly", identifying the problems encountered and the solutions devised. It proceeds to describe the general techniques developed to solve the problems of animation and suggests considerations for their implementation in a system designed to facilitate character animation. The thesis is concerned only with the use of three-dimensional, computer graphics to produce character animation. The application of computer graphics to two-dimensional animation is not discussed.

It is assumed that the reader is familiar with the field of computer graphics animation. Readers unfamiliar with the field may wish to read Newman and Sproull, or Foley and Van Dam [38,17]. "The Basic Concepts of Three-Dimensional Computer Graphics for Artists" or "Introduction to Computer Graphics" provide good, easy-to-understand, non-technical introductions to the field and emphasize issues of special concern for the artist [47,58].

The observation has been made that computer-generated animation looks too much like computer graphics, as if the animator should try to disguise the fact that the animation was indeed computer-generated. Computer graphics does have obvious, discernable characteristics. So do oil or watercolor paintings, sculpture, architecture, picture books or any other artistic medium. In the United States, any piece of animation is inevitably compared with the work of the Disney studio. It was not our goal to try and replicate the handsome drawings produced by the Disney studio for its animations. The strengths of computer-generated animation lie elsewhere. The most obvious example is the ease with which computer graphics can produce convincing representations of three-dimensional space and movement, an effect that traditional animators find particularly difficult to achieve.

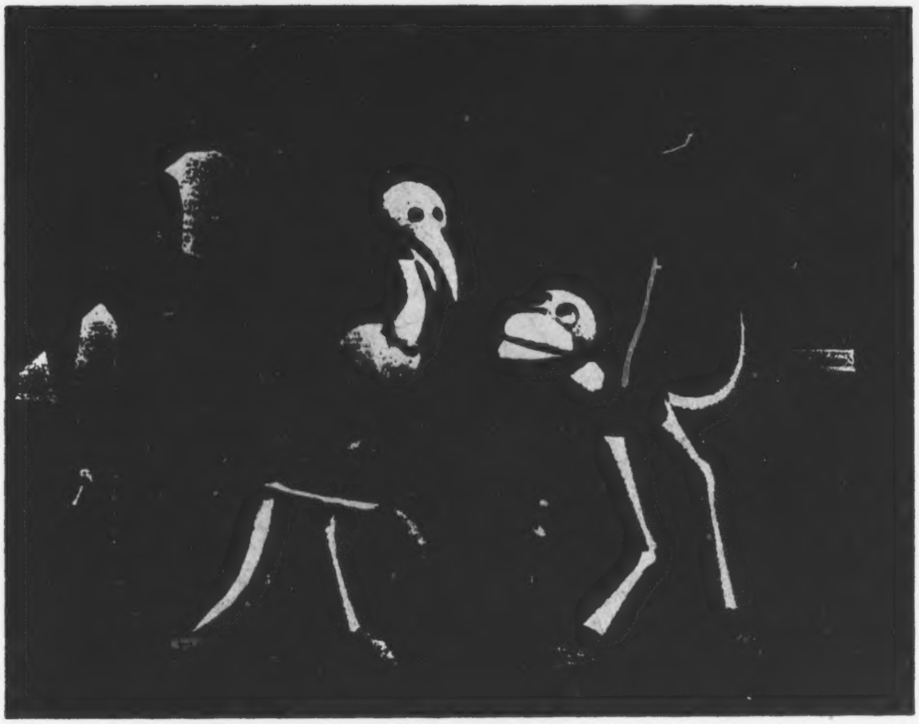


Figure 1.
Snoot (left) and Muttly (right)

CHAPTER 2

The Environment

It is difficult to discuss computer graphics, particularly the facilities at a particular laboratory, without resorting to the use of the local nomenclature. There are instances where a particular command name may be highly descriptive and convenient to use as an identifier of a very specific process. In some cases, the use of such names facilitates reading. When I use such an identifier, I will flag the new word by placing it within quotes and immediately describing it. Thereafter, it will be treated as any other noun. Program names will be treated as proper nouns.

"Snoot & Muttly" was designed at the Computer Graphics Research Group (CGRG) of The Ohio State University [55]. Computer time for raster image calculation and the use of an Electronic Still Store to record "Snoot & Muttly" on videotape were provided by Cranston/Csuri Productions, Inc., Columbus, Ohio.

The animation environment at CGRG is based upon the "animation pipeline" as described by Csuri, et al [15]. The "pipeline" consists of three stages, plus a fourth stage for post-processing:

(1) Data Generation. Creation of objects to be animated occurs at this stage. The majority of objects at CGRG are polygonal mesh structures, and it is at this stage that that polygonal structure can be manipulated.

(2) Scene Description. In this stage the objects, their transformations and hierarchies can be specified to programs that generate image descriptions.

(3) Rendering and Viewing. Several display algorithms and viewing modes are available and suited to differing purposes [13] pp. 9-18.

(4) Post-Processing. Paint programs, as well as title and buffer manipulation routines are provided.

The modular design of the software provides the animator with access to a variety of algorithms and interactive techniques at every level. A common command language and common data formats present the animator with consistency and enable processes to communicate with each other and utilize common objects [13] p. 13. Command language scripts and data descriptions can be edited by the animator outside of any program.

Two primary animation systems existed at CGRG when "Snoot & Muttly" was in production, Julian Gomez's Twixt and David Zeltzer's Skeleton Animation System (SA) [23,63].

Twixt had several advantages over SA at that time. The two major advantages of Twixt were that it was visually interactive and it allowed for the animation of several characters together [56] p. 34. Twixt was the animation system chosen to animate the film.

In its early stages, Twixt was a key-frame animation system. Given a graphic display, Twixt allowed the animator to interactively construct a scene. The animator specified objects and transformed them in one of two ways. A transformation for an object could be selected and "knobbed" in the x, y, or z axis until the animator was visually satisfied. Alternatively, the animator could enter a specific transformation command at the keyboard and the display was updated immediately. The eyepoint, center of interest, and light source could be manipulated as well. Object hierarchies could be constructed by "attaching" one object to another. Linear shape interpolation was possible between key, rigid data structures.

When the scene was composed satisfactorily, the animator had assembled one "frame" of animation. Animation at CGRG is structured on the concept of the frame, a concept borrowed from previous forms of animation. The number of frames needed for one second of animation depends upon its final format. Animation recorded on 16mm film is projected at a rate of 24 frames per second. Animation recorded on videotape is viewed at a rate of 30 frames per second.

An animation sequence is created by the animator by building a number of "frames", each of these acts as a key frame. "Key-frames" were built for every frame at which a transition in an object transformation occurred. The "in-between" frames were calculated by interpolating between key-frame object transformations. The collection of transformations for the key-frames was known as the Twixt "script". The script was organized by key-frame, listing the commands necessary to construct each.

The script was easily read and edited. This meant that the animator was not bound to the graphic display device. If necessary, a Twixt script could be constructed or modified from any keyboard and input to Twixt at a later time for visual display.

Gomez describes a major flaw of a key-frame system:

> Unfortunately, a keyframe records one state, the total state of the frame. Some objects in the scene may not have any transitions at that frame, indicating that a keyframe is necessary wherever any object has a key action, leading to a large number of keyframes. Thus objects that don't really belong in a keyframe must be pre-interpolated before capturing the keyframe [23] p. 122.

Van Baerle details the problems this causes the animator [56] pp. 15-16.

Midway through the production of "Snoot & Muttly", an event-driven version of Twixt was produced. An object's transformation, blend (shape interpolation), color and "attach" information are now known as display parameters. A key display parameter is known as an "event". The event, associated with its frame number, ease-in count, ease-out count and interpolation type (linear, Catmull-Rom, etc.), is entered into a linked list of events for that particular parameter. The event list is known as a "track" [23] p. 124. Each display parameter for each object is given a separate track. Timing of an action is easily changed by shifting the frame number of an event. The script is now a collection of all the tracks for an animation (More will be said about key-frame and event-driven animation systems in Control Methods, Interpolation).

CHAPTER 3

Making 'Snoot & Muttly'

3.1. The Story

The computer animator can benefit immensely by studying the history of animation. A great deal of the information learned by traditional animators, working with two-dimensional (2d) and three-dimensional (3d) media, is relevant to computer-generated, three-dimensional animation. The process has changed, but the end result is still an animated 2d image.

"Snoot & Muttly" was, like most current, computer-generated, character animation efforts, experimental in nature. Its goal was to use computer-generated, three-dimensional characters to tell a story. Literature documenting the experience of two-dimensional animators provided excellent advice. Thomas and Johnston's Illusion of Life was an invaluable resource [52]. It emphasized the importance of a good story, "First, there must be the big, simple idea: the story you can tell in two sentences" [52] pp. 217-18. Obviously, it ought to be a story the animator thinks is worth telling. What the animator wants to say with the story will influence how it is told. The better the characters are developed, the more believable and entertaining the film will be. Character development hinges upon the animator's ability to model the character's personality and motivations and communicate them to the audience.

3.2. Design of Characters and Environment

The story specifies some of the requirements of the character design. The method chosen to animate the character will dictate its construction and hence also its design. The design of the character must satisfy the demands of the story as well as the construction dictated by the animation system (see System Specifications, Control Methods and Modeling). The design in turn influences the character development, the possible "business" of the character, even the favorable camera angles.

Through the production of several films, Van Baerle had developed a process of animation derived from existing

CGRG software and influenced the design of newer software more suited to her needs [56]. Van Baerle had begun working with shape interpolation (see System Specifications, Modeling Schemes) as an animation technique in her first film, "Amorpha and the Fractalites". Her next films, "The Runner Animation" and "The Uneven Bars Animation" used shape interpolation to animate an entire character. For "Snoot & Muttly", she had designed more complex jointed characters with shape interpolated tubular appendages.

These appendages were the legs, tails, and necks which would be animated with shape interpolation and appear to be plastic and pliable. The appendages could easily be built procedurally with the "Tubular Object Generator" (Tog) [53]. Tog will build a tube about a 3d path of points. The user can specify differing radii at different locations along the input path and Tog will interpolate between them to give a smooth profile. Thus the legs were constructed with radii which were wide at the hip and foot, narrowed above and below the knee and widened again at the knee. The tail was wide at the base and narrowed to a very tiny radius at the tip. The neck was wider at the base and narrower at the top.

The heads, beaks, eyes, feet, and bodies were rigid objects. All pieces of the characters were effectively jointed. The rigid portions would be animated by rotation about these joints.

The paths for the tubular legs were generated from control points determined by 2d drawings of the legs (Appendix C). The leg cycles were designed from the side as 2d drawings. The movement of the legs was all within the z-plane. This method provided an easy way for the animator to design leg movement. This method created natural movement fairly easily. The appendages could be stretched or squashed at will, although the dimensions in the birds were kept consistent to maintain a sense of solidity. However, complex motions in more than one plane became too cumbersome for the animator to construct by hand (see Compromises).

The tail and neck moved in three dimensions. Orthographic projections were used to aid the drawings of their motion (Appendix C).

Although several interpolation schemes are possible (see System Specifications, Control Methods, Interpolation), linear interpolation was the only method available during the production of "Snoot & Muttly". Linear interpolation between two extremes provides straight-line

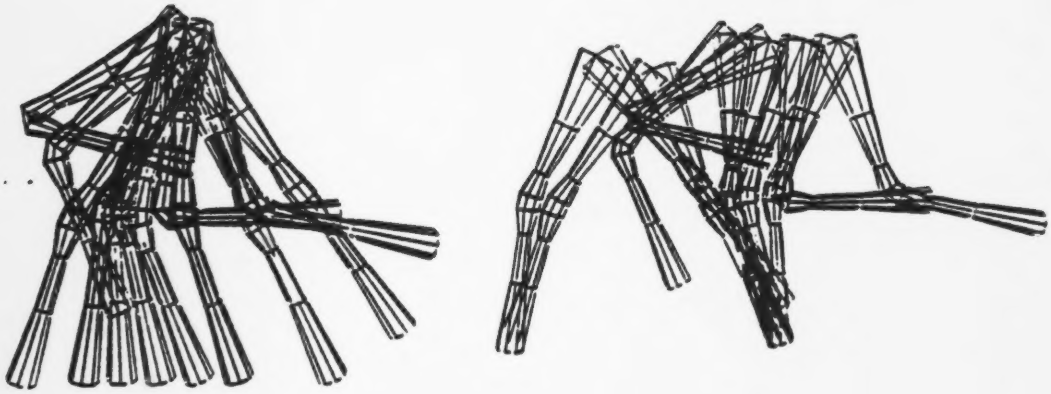


Figure 2.
Tubular Legs as Built at Zero (left)
and as Paired and Placed in a Walk Cycle (right).

movement and many key objects are needed to simulate smooth curved movement.

The following system was developed for creating leg cycles. First, the 2d walk cycle was drawn. The walk cycle consisted of "stick" drawings on graph paper of the legs in 5 to 10 key positions as they moved through one complete step. These key positions, or keyframes, were thought of as such moments when a heel strikes or is lifted from the ground, the leg begins to bend or is bent at its maximum, etc. Care was taken to ensure that the last leg of a cycle matched the first leg. Coordinates of the hip, knee and foot were taken from these drawings.

A "tube" was produced for each key leg in the cycle. The hip of each leg was placed at zero, and the coordinates were translated accordingly. (An early attempt built each leg in its final position in x. However, when "changing shape" between the last leg of a cycle and the first leg of the next cycle the translation necessary for the second step resulted in an undesirable interpolation). Leg data built for a cycle animated one leg. To animate a single leg, twixt only needed a list of the "tubes" and the respective keyframe numbers that they represented. The legs animated at this point would portray a single leg bending and stretching in place, going through its portion of a walk.

Twixt would then be given another list of the same "tubes" with keyframe numbers out of sync with the previous tubes by half a cycle. An additional command would "attach" the new leg to the old at a small distance in z equal to the width of the hips (the actual process of attaching one object to another is discussed below). In other words, the legs possessed the same x and y locations but space was placed between them in z so that they stood side-by-side. The first leg became the right leg, the second the left. The left leg was always attached to the right leg. Because the legs are built with their hips always at zero, if they were animated at this point, they would effectively be walking in place. If we want these legs to walk through "world space", we need to place each pair of legs an increasing distance from the y-axis along the x-axis. The location needed to maintain the position of the foot on the ground was the position on x that the leg occupied before translation to zero. In twixt, a placement command at each keyframe is issued for the first leg. Because the second leg is attached to the first, it will move with the first leg. When animated in a pencil test the character will appear to walk in a straight line along the x-axis. The timing of the walk can be changed by "sliding" the keyframe numbers of the tubes (see System Specifications, The Cyclical Design Process).

In actual construction, the joints for the legs were inserted deeply into the body and feet; the tail joint into the body; the neck joint was inserted into the body and head; and the beak into the head. The intersections were calculated within the scan-line display algorithm Poly_scan, by Shaun Ho, which was used for final image calculation [27]. The intersections calculated by Poly_scan were not always correct, but were usually good approximations and performed well for our animation.

Character construction consisted of rigid components joined together by flexible "tubes". The flexible "tubes" served the functions of legs, tail and neck and largely relied upon shape interpolation for their movement. The feet, body, and head were rigid objects whose movement resulted from rotation (see Character Structure).

The characters were constructed as a hierarchy of attached objects. When object A is attached to object B,

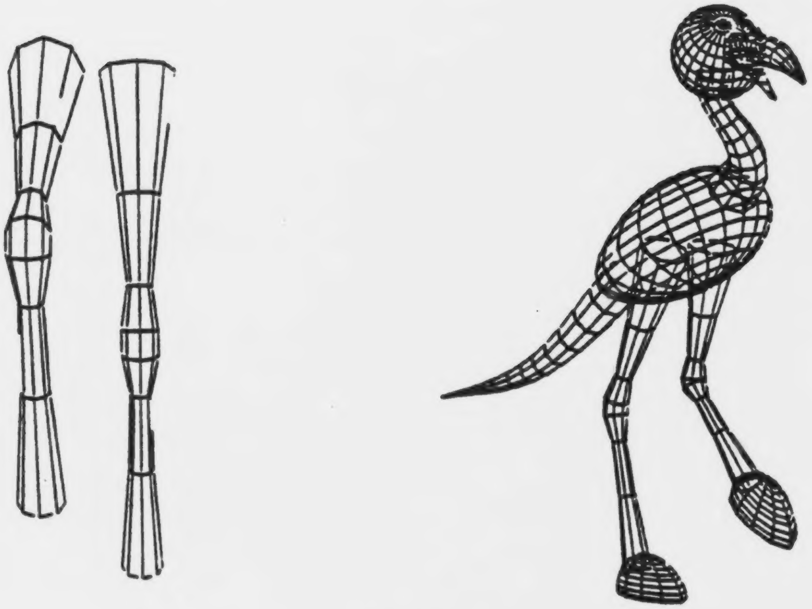


Figure 3.
Two Legs "Attached" with Displacement in Z (left) and Final Construction of Snoot (right).

rotations, scales or translations applied to object B are also applied to object A; transformations to object A do not effect object B. In other words, object A goes where B goes, but object B does not follow A unless it is reciprocally attached to object A. Object A is attached to object B at a specific point in "object space". This point is a location in object B's "object space" and has no relationship to the surface definition of object B at all. So, any change in object B's shape causes no change in object A's position. We had no method of attaching an object to a plastic object to create plastic, articulated figures.

We needed to make object A sensitive to object B's shape interpolations in order to keep the feet on the ends of the legs and the head on the end of the neck of each character. The solution was the program "Bird_attach" (see Special Purpose Programs). Object A was attached to object B in the first keyframe of a scene. "Bird_attach" calculated the displacement in x, y, and z of the attachment point of object A from a target point identified by number on object B. For each succeeding frame of the scene, the target point is found in the temporary interpolation of object B. Object A's original displacement is added to the new target point's position and object A is reattached at that point. This is what became known as a "loose" attach, only the attached object's position is effected by the shape interpolation. An example of a loose attach might be a car on a ferris wheel. As the wheel turns, the car which is attached to the wheel will go up and around. The car will not rotate itself, however, and remains in its original vertical orientation at all times.

The proper attaches were output to a "twixt" script which could then be altered, if necessary, by the animator. After Snooty's feet were attached with Bird_attach, they were rotated within "twixt" so that the toes would point as Snoot walked.

"Loose" attaches worked fine for the feet, but did not work as well for the head. When the head was loosely attached to the neck and then turned to look to the side, the effect was very artificial looking. The head looked like an inanimate ball of little substance jammed onto the end of a stick. It was evident that the action of the head was not isolated, but was related to the action of the entire body. The neck would tuck, the body would rotate, and the knees would bend in such a way as to create what artists call the "action line". But when the neck bent to the side, in a side look, the head maintained its original forward looking orientation. The head had to be rotated by hand into the proper orientation for a look to the side. For one keyframe it was a simple matter, but for an entire film it was too tedious and was impractical. To properly orient the head by hand often would require the rotation of the head about 3 axes. Looking at each keyframe the head would appear to move smoothly. But when animated, the differing combinations of rotations used in each keyframe produced unpredictable results. The rotation necessary to properly orient the head on the neck as it bent, turned and dipped seemed an easy task for the computer. So an option was added to Bird_attach. Now the new rotation could be calculated as well as the new attach point.

The bird's necks were standardized as eight-sided tubes (eight sides were the minimum number of polygons necessary to give a rounded appearance). "Bird-attach" knew the head was attached to the end of the tube that formed the neck. The last eight points were always in the plane that "capped" the end of the neck tube. The normal of this plane was found and the rotations of the head necessary to align the head's y-axis with the normal are calculated. This type of an attach became known as a "tight attach", the position and rotation of the attached object would match that of the identified polygon on the shape interpolated object. A good example of a "tight" attach would be a nozzle screwed tightly onto the end of a garden hose. As the hose bends and rotates so does the nozzle. To turn the nozzle in any direction the hose must bend. The nozzle appears to be firmly bound to the hose and moves and rotates with it. The purpose of a tight attach was to give the appearance that most of the motion of the head was a result of the bending of the neck. The attaches and rotations were placed in a twixt script which the animator was then free to access and adjust interactively for more expression.

Once the general design of the birds was finalized it was modified to give each of the characters their own identity. Snoot was given a longer tail, a narrower beak and a more erect posture to elicit the appearance of refined gracefulness. Muttly was designed for a more gangly appearance, with longer legs, shorter tail, a broader beak, and larger feet.

The movement of the characters was designed to further promote their personalities. Snoot's movement was to be kept dignified and erect. Snoot's head would be held high and toes were to be kept carefully pointed. As the story progressed, Snoot's movements were to become more and more animated, less confined. Muttly's natural posture was kept at about the middle of the total range of movement. From this position, Muttly could easily stoop to sniff the ground or a stretch to peek in a window. His movement was to be quick and spontaneous.

Color also was used to help define their personalities. Snoot's bright red attracted attention, yet it was conservatively combined with browns and oranges. Muttly was decked out in bright blues and brilliant yellows.

The set was designed to provide a fantasy look. Randomly colored houses were set in the midst of a forest of fanciful vegetation. For more detail about the environment and the construction of the characters, see Van Baerle [56] pp. 35-42.

3.3. Animation System and Character Design Test

A walk cycle was produced with the finalized Snoot design to test the animation "system" and character design that had been developed. The walk cycle also tested the functionality and appearance of the character in motion. The success of the animation depended a great deal upon how the characters appeared as they moved.

3.4. The Storyboard

After a walk cycle had been successfully animated, the potential of the system was better understood and work on the storyboard could begin. The story was first enacted "live" and filmed in an attempt to establish tentative timings, spatial relationships and camera angles.

A great deal of time was spent in planning the storyboard. At this stage, we were purposefully looking for ways to increase the expression of our characters' personalities. Muybridge was consulted for the leg and neck action of birds as they walked [37]. Multiple sketches of actions, situations, possible "business", expressions, and character poses were produced. Those that promoted the story and the character's personalities were incorporated into the storyboard (Appendices A and B).

An emphasis was placed upon keeping the story, motivations, and actions simple and clear. Every scene, every shot and action was evaluated in terms of its contribution and anything that wasn't necessary to advance the story or the characters were cut. And, because computer graphics is time-consuming and costly, we looked for places that we could cut simply to save time and calculation. As the project progressed, some scenes could be combined and some were deleted entirely. Some reaction shots were added for better communication. The revision of the storyboard was a continuing process throughout the production.

We quickly identified two problems that needed to be solved to produce the animation. The first was getting Snoot and Muttly to walk on curved paths. The second problem was that of the formation flying required of the bubbles. Pathwalker was written to solve the first problem, and Bubble_flight solved the second (See Special Purpose Programs).

While the above two programs were being developed, the different walk, run and skip cycles needed for the storyboard were being constructed.

Various functions of the animation system were identified that could be performed by small programs.

A program written by Donkin, "Dime", reduced the number of points necessary for the animator to specify for the construction of a leg to those of the foot, knee and hip [16]. It output the additional positions for the narrowed radii above and below the knee. These five points were then given to Tog to build the leg.

In the UNIX environment, the user can create "shell scripts" to perform sequences of programs that are run frequently. It was often necessary to run an animation "script" through a succession of ten or more separate programs. A shell script would handle the repetitive processing of scripts through a series of programs thus saving the animator a great deal of time. The shell script also ensures that no mistakes were made in the processing, that no step was forgotten or processed in the wrong order.

3.5. The Animation Process

In this step, the actual animation of the storyboard was produced.

3.5.1. Design of the Motion Paths Because the set was set entirely upon a 2d plane and the majority of Snoot and Muttly's movement was on the plane, their paths through the set could be described as 2d curves. A map of the set was drawn and used to determine the paths of the characters. The characters' choreography was based upon the dimensions of the set. The houses and trees had exact locations which had been revealed to the audience in the opening camera move. We felt we had to remain true to these to maintain a feeling of physical space. Therefore the characters had to be maneuvered so as not to necessitate the relocation of houses or trees (Appendix D).

The storyboard had been worked out in rough sketches. As the storyboard was refined, we began laying out the paths of the characters on the map of the scene. The walk cycles were mostly complete at this point and we knew how far each character travelled with one step of the each cycle. The construction of paths was a carefully juggled relationship of speed, distance and time. If Muttly was to walk for three seconds it was a simple matter, knowing that Mut took two steps per second at one unit apiece, to calculate that a path six units long was needed. A path six units in length would be crudely measured on the map with a flexible ruler (The choreography of two characters with specific step lengths and differing periods of motion and rest is discussed in more detail in Compromises).

Vertical motion of the characters was sometimes a problem. Often, vertical motion required no supporting legs or possessed no movement while elevated that would necessitate a calculation to maintain a foot location on the ground. Such movements were vertical jumps and leaps during running or standing on tiptoes. The vertical motion involved could be easily specified in the vertical placement of the legs in the script. Pathwalker would simply pass these positions on to the output script.

A problem was encountered when Snoot was elevated on the front step. Snoot was required to walk at this elevated level. Pathwalker recognized supporting legs by the distance of the ankles from the ground plane, and the level of the porch elevated the legs beyond the support range. So Snoot was animated on the ground plane and translated upwards for those portions. Another approach would have been to change the range within Pathwalker.

The main bubble path was also drawn on the 2d map and was often interelated with the "bird" paths. A side elevation was used to determine the height of the path along it's route as the bubbles flew over or around the birds.

3.5.2. Selection of the Camera Angles

Eyepoints are chosen tentatively in the storyboard stage. Once the characters' paths and positions are located on the map, the eyepoints can be solidified. The line of sight can be drawn between the eyepoint and center of interest. Originating at the eye point, the view cone straddles the light of sight and opens towards the center of interest. Our view angle at the eyepoint was 45 degrees. The view cone, as drawn on the map, will contain everything that will be visible on the screen and can even provide an approximate idea of an object's horizontal screen location. A template of the view angle will accelerate the process.

As the animation is set up in Twixt, the appropriateness of the view parameters becomes apparent. If an eyepoint or center of interest is found to be weak, a replacement is entered and the resulting view of the scene is rendered on the screen immediately. Several alternative eyepoints may be tried until a preferable one is found.

It is not necessary to choose a camera angle before animation is begun. However, if the camera position is known, the staging of the characters can be better designed.

When "pencil testing" a segment of animation, it is wise to check how well the cuts between differing camera views will work. A camera angle may not work in the context of the previous and following camera angles. If this is the case, an alternative may be selected as described above.

3.5.3. Construction of the Leg Animation Script

A twixt script is assembled for the leg animation of a scene using information gathered from component leg animation twixt scripts (the leg cycles or special purpose moves previously assembled). Positions accumulate along the x-axis as one cycle or move is tacked onto the end of the next.

Transitions necessary between different leg cycles are set up from 2d drawings in the same manner as the cycles. Non-cyclical special purpose movement, such as Snoot's "retreat", may also be set up at this time. If the movement will be on a path, the movement should be set up first to provide the length necessary for the path. Twixt provides vector playback for motion testing. Each segment of movement is then checked in low resolution, real-time, raster playback. (see System Specifications, Low-Resolution, Real-Time, Raster Playback).

After an entire sequence of motion has been constructed and looks good in vector playback provided by Twixt, the entire motion is checked with real-time playback of low-resolution raster images.

3.5.4. Placement of the Legs on the Motion Path

A twixt script was generated by bird-attach that attached the body to the top of the right leg and the feet to the bottoms of their respective legs. The twixt script and the path of movement for the character were input to Pathwalker. The Pathwalker-generated, twixt script was checked on the Megatek (a vector display device). The character should no longer walk in a straight-line parallel to the x-axis, but should rather follow the input path. The feet were checked for erroneous movement while they were placed upon the ground.

The script output from Pathwalker changed the hierarchy of attaches to one more suited to walking in 3d. The left leg and body were no longer attached to the right leg. Instead, the body was attached to an invisible point at its center called "tiny". Both legs, positioned equal distances in opposite directions to the side, were also attached to "tiny" and hung below the body.

If the entire character, legs and body, were to be rotated to orient it on a path, then the rotation of "tiny" would do. Because the legs were attached to "tiny" and not the body, the body could be rotated forward and backward or side-to-side without rotating the legs as well. This should have been our original hierarchical structure.

3.5.5. Animation of the Head and Neck

The necessary keyframe necks were designed and generated. Because the movement of the neck was not to be a flat, 2d movement but would actually twist in 3d, it was necessary to draw the necks in orthographic projections. and input to Tog to build each neck. The tubes were then given to twixt with their respective keyframe numbers. The timing was checked with twixt. The neck was attached to "tiny" at the front of the body. The motion and timing of the neck were checked with that of the body and legs. Like the legs, the timing of the neck could be adjusted by "sliding" the keyframe numbers of the tubes forward or backward in time.

Once the timing of the neck with the body was perfected, the head was attached with Bird-attach. The head was attached in a slightly different manner from the other appendages. Instead of attaching the head to any specified point, the head was automatically attached to the calculated center of the top profile of the neck and oriented to match the orientation of the last segment of the neck.

Two keyframes, in which the head was rotated about 3 axes, may look similar but may in fact have vastly different rotations about the 3 axes. When twixt interpolates between the two keyframes, the resulting rotation is wildly unpredictable. The solution was to give the head a "double joint" to help order the rotations. Instead of rotating the head in one step about 3 axes, it is rotated in two steps, the first about y, the second about x and z. That was accomplished by attaching the head to tiny at 0 0 0 and tiny to the neck in turn. This allows all rotations about y to be performed with tiny first. The remaining rotations about x and z are performed by the head.

The head and neck motion were the most commonly used in interactions with the other characters. For this purpose, the necessary characters were combined in Twixt and the interaction checked. In this fashion, the position of the head could be adjusted to ensure that one character was looking at another. Timing on the neck and head rotations could be adjusted from within Twixt with all characters in proper relationship. Low-resolution raster images were then calculated for a final motion test.

3.5.6. Attach and Animate the Tail

The tail was designed 3-dimensionally, in a manner similar to the design of the neck. Cycles were constructed for differing tail wags. Tails were selected from these tail wagging cycles or built for special purpose motion and entered into the character's twixt script. The tails were attached to "tiny" at the back of the body. Timing was usually checked with twixt playback. If needed, a low-resolution, raster, playback was calculated for additional verfication.

3.5.7. Attach and Animate the Eyes

The eyelids, pupils, and eye whites were constructed upon successively smaller spheres that were nested within the head and attached to it's center (see Van Baerle for illustrations) [56] p. 39. Although these spheres never actually intersected one another, their close proximity often confused the display algorithm. The resulting image would render an inner sphere partially in front of an outer one. For example, the whites of the eyes, a hemisphere below the surface of the head, would "bleed through" to the surface of the head. Furthermore, the sorting of the various layers changed as the distance from the eye to the head changed. This problem demanded that the calculation of the head and the eye structures be checked for intersection problems prior to calculation of each scene. The head and its eye components would be calculated at their closest and farthest positions from the eyepoint for a particular scene. If intersections were evident, the scale of the different components would have to be scaled smaller until the intersections disappeared. The distance needed between the components of the eyes, to maintain acceptable sorting, is evident when one of the characters turns its head to the side.

The pupils were rotated interactively with twixt. The pupils were separate from the eye whites and were animated individually. Low resolution, raster tests had to be calculated for motion tests as the data was too complex to discern identifiable parts of the eye in twixt's vector display.

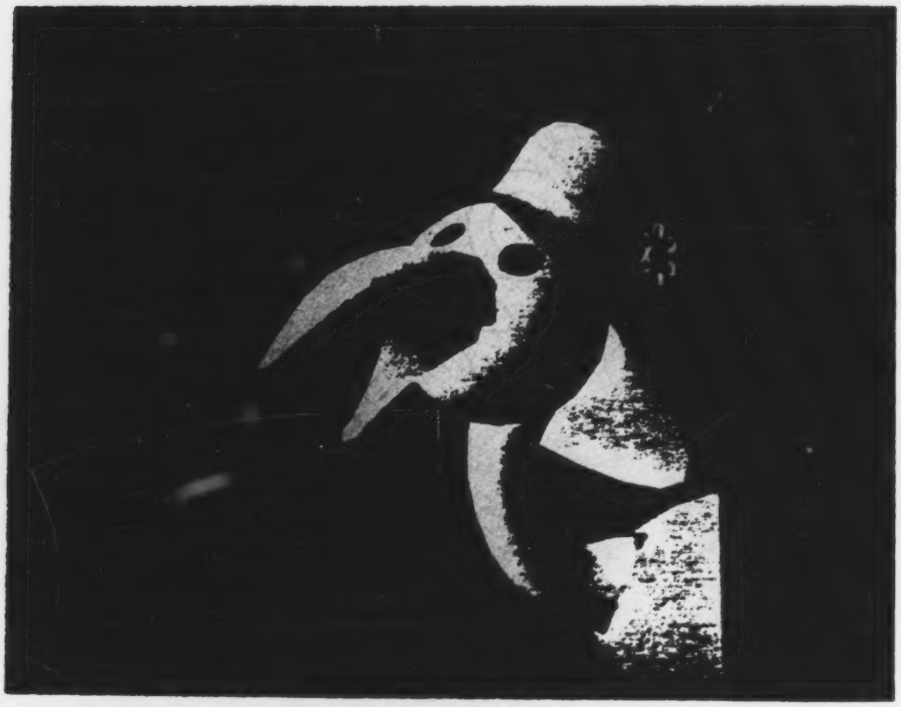


Figure 4.
Close-up of Snoot's Head

Because the eyes are roughly on the sphere of the head, the perspective on each eye can be very different as they move to the side of the head, especially when head is between a three-quarter view and a full profile. This results in the character looking slightly cross-eyed. To compensate, the eye furthermost from the viewer, is rotated back towards the viewer slightly. The same is done to the eyelids to reduce the edge of the eyelid visible through the eye socket.

3.5.8. Animation Considerations

The characters, excluding the bubbles, were generally animated separately. However, when the scene called for physical interaction between two or more characters, it was necessary to load the characters involved into twixt where all the characters could be interactively viewed and manipulated. This was necessary in several circumstances:

to check the character's relative positions; to check that the characters weren't walking through each other; to focus the eyes of one character upon another; or to check the reaction timing of one to another. The characters could be combined at any time, but were generally combined at the earliest appropriate stage, with as few appendages as were necessary to give the needed information. The less data present in each character, the faster the calculations and thus an increase in response time and playback speed. After the adjustments were made, the twixt script was output and edited to produce scripts for individual characters again.

Simplified data was occasionally incorporated into the character structure to speed up twixt response and motion test calculations. A simplified ball, constructed of 24 polygons, could be substituted for a detailed head of 250 polygons. If it was necessary to study the orientation of the head, a simplified beak was added. The body and feet, constructed of rigid objects like the head, could also be replaced with very simple rough approximations. At final calculation time, the simplified portions were replaced with the detailed objects.

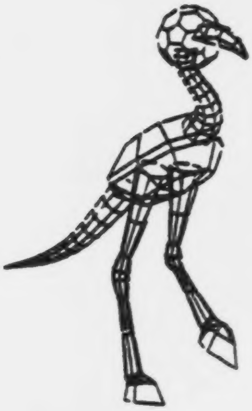


Figure 5.
Snoot Constructed from Simplified Data

If it was not necessary to see a particular body part, twixt provided the option of turning it "off". The object would no longer be displayed, but its presence was still accounted for in the hierarchy. If it was necessary to study only the movement of the neck, the rest of the body could be turned off. The influence of the other objects remained. They were merely invisible. The neck could then be animated within the scene by itself. This also increased response time.

A third conservation measure, to save display device memory, involves an option to display only the points at the vertices of the objects' polygons rather than drawing the polygons themselves.

"Snoot & Muttly" did not use a lot of "squash and stretch" of form. It did, however, make use of overlapping action to give a sense of "squash and stretch" through the action. An example of overlapping action is found in the following sequence of a step: A foot hits the ground; the knee waits a couple of frames before it bends; the body begins to rotate forward a few frames later; and the neck and head each follow after a few more frames. Overlapping action spreads the movement of individual parts across several frames and gives the feeling of weight and inertia to the figure.

Although each appendage is animated separately, it is important to recognize that they are all participatory in even the slightest action of the character. Acting out a task can help the animator identify the important joint or body movement that characterizes a movement or attitude. Action sketches of the character at various points in a movement can also help the animator determine the role of each appendage in defining the motion. These exercises can be very helpful when the animator is assembling key-frames or events of individual appendages.

3.6. Image Calculation and Storage

The rendering program maintained the coverage of each pixel and saved it with the raster image (see System Specifications, Rendering Considerations, "Cel" Capability). The resulting images where then manipulated in a fashion analogous to traditional animation 'cels'. Images with coverage were composited to produce one image from several separately calculated images.

In general, Snoot and Muttly were calculated separately, each on their own 'cel'. The background was usually rendered in two or three 'cels' (one each for foreground trees, houses, and background trees and ground). The bubbles were calculated in as many as three separate 'cels', depending on their interaction with Snoot and Muttly.

The production of each character as a separate image was convenient for a number of reasons. The calculations necessary for one scene could be conveniently split into those necessary for each character and calculated on different machines. This would speed up the turn-around time necessary to view a particular scene in final form. It also avoided recalculation of all characters if a mistake involved only one. Finally, for scenes with a still camera, the background was only calculated once. Several character frames were layered upon the same background image.

3.7. Video Production

The 'cels' were composited and recorded on an Electronic Still Store (ESS). The ESS recorded the frames in NTSC format (Television code standard used in the United States) on one of three disks. Once a sequence was placed on the ESS, it could be played back at full resolution in real-time. The total capacity of the ESS was nearly one and one half minutes of animation. With this capacity, as each scene was completed, it could immediately be viewed in the context of the animation immediately preceeding it.

Any problem sequences could be readily identified. If the problem involved only one character, the 'cels' for that character were recalculated and recomposited, and the new frames were simply written over the old on the ESS. It was not necessary to recalculate or re-record the entire scene, only the faulty frames needed to be replaced.

If a cut between two camera angles was not satisfactory, one of the scenes or a transition scene would be recalculated from another viewpoint. This involved no more trouble from the animators than using Twixt's interactive vector display of the scene to choose a new vantage point. Once a new camera angle was selected, it was a simple matter to edit it into the script. Of course, all of the cels had to be recalculated from the new eyepoint.

If a character got "lost" in front of the background scene due to lack of contrast between the character and the background, post-processing functions were used to lighten or darken the background or foreground image before the images were composited. Otherwise, the lighting for the character or the background was changed and just that element recalculated.

If the leg was found to be supporting, but was not a supporting leg in the last frame, it was identified as a new supporting leg and the absolute position of its bottom point and the absolute rotation of the foot were found and stored. Script commands were output to "attach" the leg to the body.

If the supporting leg was not a new supporting leg, then it had to be rebuilt to maintain the same absolute foot position. The bottom point had to be moved to maintain the same absolute location as the bottom point of the leg in the previous frame.

The bottom point of the leg had to be moved to the absolute position of the bottom point of the leg in the previous frame. The displacement between the current position of Tiny and the last position of the bottom point of the supporting leg was found. Tiny was translated back to the origin, carrying the bottom point with it and retaining their relationship. Tiny's rotation was then undone. The leg attaches and transformations were undone. This returned the bottom point to its position relative to the leg with the hip placed at 0 0 0 with no rotations, the position of all the shape interpolated legs.

The new bottom point rarely fell in the plane of the original legs. The rotation about the x-axis required to return the new bottom point to the plane of the original leg was found. All control points of the original leg for the current frame were rotated about the x-axis into the plane containing the hip and the new bottom point.

The bottom point of the original leg was replaced with the new bottom point. The new tubular leg was built from the new control points and substituted for the original leg in the Twixt script. When the new leg was attached to Tiny and Tiny was transformed into position, the bottom point of the leg had the same absolute location as it did in the previous frame.

4.2. Bubble Flight

The problem involved the animation of several bubbles in such a way as to allow them to be controlled as a group, and yet have some random movement as individuals. The problem was solved by establishing a 3-d grid which was placed and oriented along a path. The dimensions of the grid were variable, as were the number of bubbles. Bubbles occupied the nodes at the intersections of the grids. The nodes were spaced at 2 * radius of bubbles on all axes. This distance packed the bubbles reasonably densely and reduced the chances of collision among bubbles moving about neighboring nodes. The ratio of bubbles to nodes was roughly 15 to 1. This also affected the density of the bubbles and left enough nodes empty to provide an impression of randomness of placement among the bubbles and mask the grid positions.

The control points of the general path that the group of bubbles was to follow were input. The dimensions of the grid, number of bubbles, and the percentage of the bubbles that were to move each frame were specified. The output consisted of a file of control points for each bubble's path.

For the first point, the 3d grid was built and an alignment vector was attached. The grid's origin was

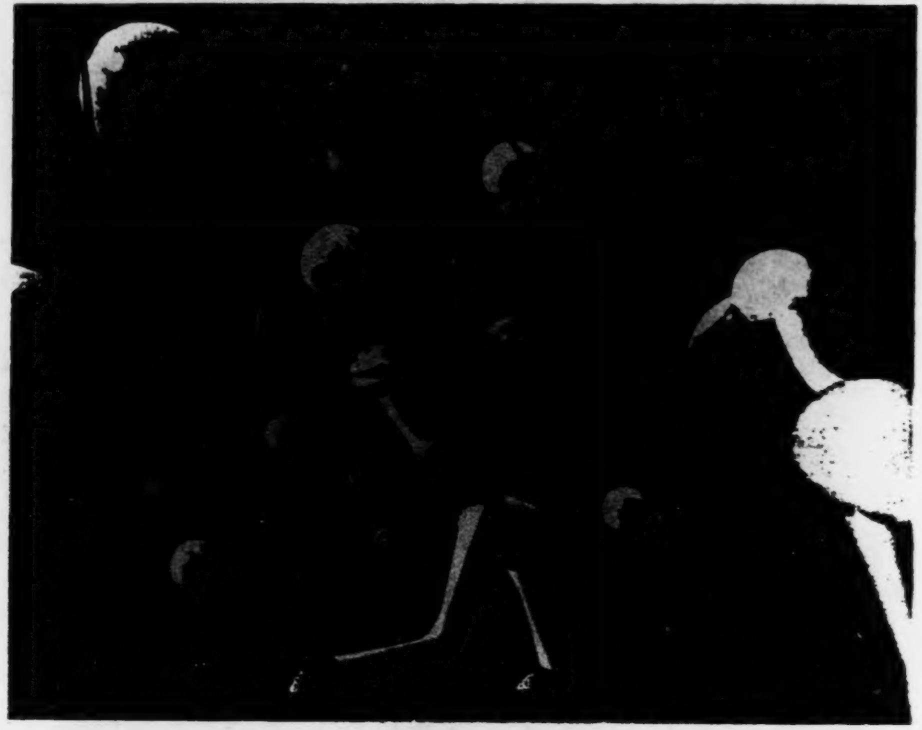


Figure 6.
Muttly (left) and Snoot (right) with Bubbles

placed at the first point of the path. The grid was oriented so that the alignment vector was directed at the next path point. The bubbles were randomly assigned to the nodes of the grid. A file was opened for each bubble and the coordinates of the node assigned to each bubble was placed in the file for each.

For each succeeding point, the grid was moved and oriented. The bubbles were randomly selected to move to a randomly selected neighboring, non-occupied node or to remain at their currently assigned node. The coordinates of each bubble was appended to its file.

Afterwards, each control file was run through a series of programs:

(1) Tog, to generate the actual path from the list of points in each bubble's file [53].

(2) Penbldr, to construct the twixt file commands necessary to build a keyframe representing the bubble's position at each point on the path [28].

(3) Flightsim oriented the bubble properly along the path.

(4) Pen_combine combined the singular Twixt files for each bubble into one Twixt file containing the animation of all bubbles.

(5) Bubble_check read the final Twixt file and checked for bubble collisions. Collisions were possible because Bubble_flight was not run for every point in final path but only generated control points for each bubble's path. Actual collisions were rare.

CHAPTER 5

Compromises

The system developed for the production of the film was constructed primarily from software already in existence at CGRG. As mentioned above, some of the software was modified to accommodate the shape interpolation approach and a few special purpose programs were developed. But the system, far from an ideal character animation system, required a number of compromises.

A major compromise was the necessity of animating the character one element at a time, first legs, then feet and body, then neck, tail, and head, and finally, the eyes. Of course, the animator could go back and adjust a previously set appendage. But the animation of a character is not composed of seperate actions of the appendages. Each movement involves the entire body. The smallest movement of the head may be supported by a shift in weight at the hips, and a bend of the neck or a leg. Our system demanded that the animator have an idea of a total body posture and manipulate each appendage into its proper location. Quick "action" sketches, portraying the character's entire pose, helped the animator to maintain an image of overall body posture.

Many moving camera shots had to be replaced with still camera shots because of the computational time required by the background. It was much too expensive to calculate the background in every frame. So the background was generally calculated once per shot, and the characters were calculated separately. The characters and background were composited to produce the final image.

Path planning in the storyboard stage required a lot of manipulation to achieve the desired timing and staging. As a case in point, consider the following scene. Snoot and Muttly are standing side-by-side. Snoot walks away. Muttly watches for a moment, then chases after and catches up with Snoot. The timing from the storyboard gives Muttly 2 seconds to watch Snoot and 3 seconds to effect a rendezvous, or 5 seconds for the entire scene.

The cycles have been completed and we know that Snoot takes 2 steps per second. Each step is .5 units in length. Muttly's running cycle consists of 3 steps a second of 1 unit each. Snoot will have moved a total distance of 5 units in 5 seconds of motion. Muttly will have traversed 9 units in the last 3 seconds of motion. Muttly will have quickly passed Snoot and will finish the scene 4 units ahead. The intent is to have Muttly reach Snoot's side at the end of the sequence.

One solution to the problem involves the design of the paths. Snoot's path, the shorter of the two, is drawn to be more direct. Muttly's path is less direct and wanders more. Snoot's path can also be designed to turn away from Muttly, causing Muttly to run even farther to move around to the side of Snoot.

Another solution might involve changing the timing. If Muttly gives Snoot 4 more seconds head start, they will both require paths 9 units in length.

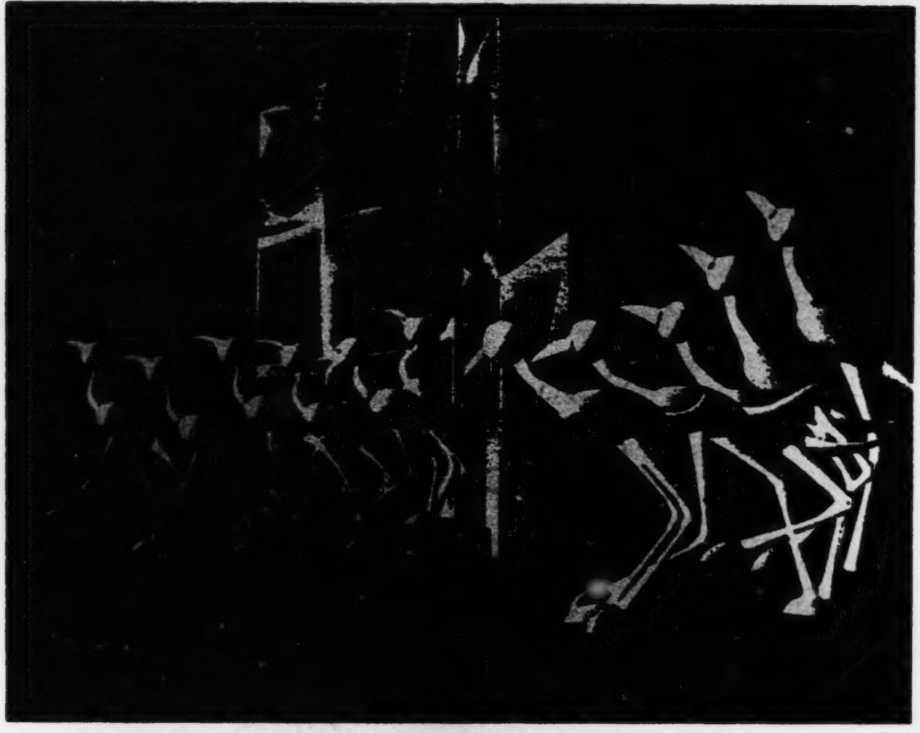


Figure 7.
Elapsed Frame Showing Consectutive
Positions Necessary for Movement

If it is important to maintain the timing, editing might supply the solution. A camera shot that makes Snoot's and Muttly's positions briefly ambiguous permits a cut back to another shot that re-establishes new positions that are more advantageous to the desired timing. For example, immediately after Snoot steps off, the camera switches to a close-up of Muttly. The close-up may be maintained until Muttly takes a running step or two. The audience will assume that Snoot has continued walking but will possess only a vague idea of Snoot's current location. We take advantage of the uncertainty to position Snoot forward 4 units and cut back to a two shot to show Muttly catching up to Snoot.

This method works better if the characters are not near any landmarks. A small change, or even a large change in a character's position is much less noticeable if there are no landmarks nearby.

There was no independant control of speed and path for the bubbles; each point defining the path was a point in time. Speed was varied by shortening or lengthening the path or increasing or decreasing the number of points in the path until the bubbles moved as far as necessary or as fast as necessary. The bubbles couldn't speed up or slow down in flight.

One attempt at isolating path from speed generates more points than necessary to describe a curve. An algorithm can then be used to determine the changing increments of points necessary to vary the speed over time. A function must be available to place the points equidistant along the path to ensure accurate motion.

A better method uses arc length parameterization [25,49]. Using arc length parameterization, the exact location of a point at a specific distance along a path can be found. It would be possible to describe a speed curve, input it and the space path and have the program output the successive locations (see Motion Control, Motion Functions).

Snoot's body was originally colored a pure red. It was discovered that pure primaries cause a saturation problem on videotape. The red was neutralized somewhat by added small amounts of the remaining primaries.

A section of Muttly's dance had to be eliminated. Muttly was to execute a series of pirouette-like turns.

These turns required the animator to manipulate the legs in three dimensions. Our system described leg movement from the side as a two-dimensional motion. Pathwalker would then lay the two-dimensional cycle along a three-dimensional path. The turn was too complex to be described easily in two-dimensional drawings or in three dimensions by constructing key body poses. Clearly, an algorithmic solution was necessary (see Control Methods, Simulation Control).

In one scene, Snoot turns his head away as Muttly approaches him. Muttly was to have "rubber-banded" his neck around Snoot's back to position his face directly in front of Snoot's, in a manner similar to early "rubber-hose" animation. "Rubber-hose" animation characters typically had extremely flexible appendages. There were no specific joints and the entire appendage was bendable in a snake-like fashion, like a rubber hose. The appendages were squashed and stretched drastically along their length for effect. Muttly was then to walk around Snoot's back to "catch up" with the neck. This would have involved a problem similiar to the moving-foot problem, where we identify a point in absolute space and connect it to a point on a moving body with a procedurally generated object, the neck. This move was eliminated because it violated the loosely defined physical properties established for the characters.

CHAPTER 6

The Character Animation System

The character design, the story, and the animation system are interdependent in an animation production. The type of motion possible is constrained by the construction of the animation system. The type of motion available will in turn determine the approach taken by the animator to animate the story. The design of the animation system will limit the possible character designs.

The characteristics of good story and character design are adequately discussed in the realms of traditional animation literature. The fact that our character animation is generated by computer doesn't negate this previous experience. However, the design of the animation system will bear heavily upon the design of the characters, the character of their motion and their possible actions. The computer animator must possess an understanding of the animation system to be used, its limitations and its potential. This understanding is crucial for successful design and execution of computer-generated, character animation. As the animator's understanding grows, One might hope that the animator can play an active role in the development of the animation system.

The animation of "Snoot & Muttly" not only provided an opportunity for the exploration of the animation systems available at CGRG but raised an interest in the functions of an ideal character animation system. The remainder of this thesis provides a survey of animation techniques and functions developed in the field of computer graphics. The potential of these techniques for character animation is evaluated. Special considerations for the implementation of these techniques into a character animation system are suggested. It will not attempt to discuss any specific hardware or software but is concerned primarily with the concept and design.

CHAPTER 7

Control Methods

Motion control, how the animator will direct the action of the objects and characters, is one of the most important issues in animation system design. Several solutions to the problem of motion control have been developed to successfully produce animation. In disussing the control of movement, it is convenient to use a structure utilized by Norman Badler [2] p. 28. Badler organizes motion control techniques into five categories: Key-frame (or what I will refer to as "Interpolation" for reasons which will be become apparent later), Movement Functions, Goal-Directed Behaviour, Constraints, and Simulation. Each method has its own strengths and weaknesses which make it valuable for a particular movement problem.

7.1. Interpolation

Interpolation animation systems have their roots in the traditional 2d cel animation technique sometimes refered to as "key-frame", or "pose to pose", animation. Two-dimensional animators typically produce "key" drawings in which the character is posed in the extreme and important positions of an action; these are sometimes referred to as "key-frames". "Inbetweeners" would then work from the "key-frames" to produce the intervening frames.

Instead of producing drawings of the extreme or important positions of an action, the 3d animator typically specifies parameters to position the character in the "key" poses of an action. The animation system then calculates the parameters of the intervening frames. Parameters typically control an object's scale, rotation, and placement for a particular pose. They can also be used to control an object's color and transparency, an object's "attachment" to another object, shape interpolation, and the camera's placement, rotation and view angle. It is important to note that all motion control methods can be applied to these non-movement parameters, as well.

The simplest means of implementing the interpolation concept is commonly known as computer-generated, "key-frame" animation. The animator organizes all of the objects within each key frame. The animation is a process of interpolating all parameters of one key-frame to the next. Each key-frame is a complete description of the scene; every parameter must be specified.

The difficulty with a key-frame system arises from the fact that not all parameters will normally reach their extremes at the same time (within the same frame). The strict "key-framing" method requires the animator to account for the parameters for all objects whenever any one object has a "key" pose.

The problems associated with having to account for all objects in each keyframe can be alleviated by a "multiple track" system. It is not necessary to represent every character in each "key-frame". The system maintains a separate "track" for each character as described by Fortin [18] p. 181. This allows the parameters for each character to be set and interpolated independently from the parameters of every other character. In other words, every character has its own set of keyframes. Multi-track systems are analogous to the 2d animation practice of rendering each character on a seperate cel. All of the cels for one character would represent one track. The animation of one character can be completely independent from that of every other character.

However, all parameters for a character must be specified in every key-frame of that character's "track". The animator is still faced with the fact that all parameters for a single character do not conveniently reach their extremes at the same time. Traditional animators were quite aware of this fact, and actively avoided simultaneous extremes. They stressed the importance of "overlapping action" and "follow-through" [52] pp. 59-62, [59] pp. 60-63.

This independence of parameters is gained within Gomez's Twixt, an "event-driven" animation system which extends the concept of "multiple tracks" [23]. A separate track is maintained for each parameter. It is not necessary to account for all of the parameters of a single character in a keyframe. "Events" of a parameter are set only when that parameter changes. Each parameter's "events" are independent of all other parameters. Tracks can be structured at the level of detail required. For instance, a single track for scaling can be further differentiated into three tracks, one each for the x, y, and z scaling components (Twixt will be discussed further in System Specifications, Multiple Control Methods).

7.2. Movement Functions

Badler discusses movement functions in the following:

> There are two approaches to defining movement by functions of time which may be interpreted on a frame-by-frame basis. The first involves specifying the path of an object point over time. This is the basis for the "p-curve" technique employed by Baecker [4] and extended to three dimensions by Csuri [14]. The second approach is based on geometric transformations, such as translation, rotation and dilation (scaling), and possibly varying the velocity as a function of time to modify otherwise linear dynamics [10,36,50].

The abstraction of motion as a mathematically defined path is a common implementation of movement functions. The animator chooses key locations along the path to function as control points for a curved path that is generated by any of a number of spline techniques. The points that define the path are often used to control the velocity as well as the path of an object. An object may advance on point along the path for each successive frame. This method is clumsy in that the entire path must be redefined to change the velocity of the object at any point.

It is more convenient to separate the speed and travel specifications. The object's path is defined as a 3d space curve. The velocity can be specified numerically as a 2d graph of distance versus time. A visual representation of such a "speed curve" provides the animator with a mathematical representation of the object's velocity that is intuitively understandable. The "speed curve" can be interactively designed, manipulating a few control points to produce a spline curve.

The motion along the path then becomes a matter of moving a specific distance for a specific frame, determined by the "speed curve", rather than advancing a uniform number of disparately spaced points. Distances along a curve can be measured accurately with arclength calculations [25,49].

The abstraction of an object's velocity as a mathematically defined path also facilitates the development of velocity functions that implement common velocity changes established within the 2d animation environment. These velocity changes are generally refered to as "ease-in" and "ease-out", or "slow in" and "slow out" [52] p. 62. "Eases" provided smoother motion by slowly accelerating into or decelerating out of a movement. They can provide the 3d animator with smooth transitions between rest and movement or between differing velocities. The animator should be free to specify the transition points and beginning and ending velocities, where appropriate [54] p. 3.

Badler finds path controlled motion useful for manipulating global movements of objects, but difficult to use to control movement of individual appendages of an object. He suggests that "Coordination would be extremely difficult to specify strictly in terms of such path functions" [2] p. 28.

Another common form of movement function specifies motion in terms of rotations of joint angles. Positioning the end of an appendage, given the rotations and locations of its joints, is refered to as the forward kinematics problem [19] p. 3. Badler finds the weakness of controlling motion by controlling the rotation of individual joints one of intuition: "it is very difficult for a human reader to grasp human movement strictly in terms of a composition of joint rotations" [2] p. 28. Movement functions offer the advantage of specifying movement by supplying a few commands to a program. Often, it is much quicker to define and refine movement by adjusting a few parameters than by trying to reconstruct the movement "by hand" with a key-frame system.

7.3. Goal-Directed Control

A goal-directed motion control system removes the responsibility for manipulating individual parameters from the animator. Motion is described by the animator at a higher level of interaction. Examples of such description include the specification of coordinates for specific locations to be achieved at a specific time or commands to perform named tasks. The actual manipulation necessary is determined by the system and hidden from the animator.

In Zeltzer's Skeleton Animation system the animated character possesses knowledge of its own structure and the manipulations necessary to perform simple tasks, like walking and jumping. He describes adaptive motion, which "refers to the fact that the animated character is sensitive to its internal states and its relationship to the environment" and modifies its behavior accordingly [63] p. 118.

Badler describes the representation of movement information in a goal-directed system in terms of directions, rotations, facing, shape and contacts. Directions are given to the system as a point in space or as a path parallel to a given direction. Rotations require a fixed angular change about some axis and consist of joint rotations, rotations of the entire body, and the twisting of body segments. A given facing determines the desired direction of the perpendicular to some body surface. Shape describes the path of a body joint through space or, in a manner similar to the key-frame interpolation, the configuration of the body joints to be achieved. Contacts describe a variety of relationships between one body and another or between a body part and the environment (near, far, above, support, slide, grasp, brush, contact) with a specific achievement time [3].

Zeltzer highlights two important issues involved in producing smooth animation of a sequence requiring several movements. The system must know the preconditions necessary to execute each specific movement and the system must produce smooth and natural transitions between one movement and another.

> Preconditions might require that a certain skill be ongoing with or executed immediately before a second skill; or preconditions may specify a given body posture and/or various relations between the figure and the environment ... Transitions must conform to kinematic data, if it exists for a given movement, or conform to everyday observations of moving figures [63] p. 113.

Within a goal-directed environment, the animator must be capable of describing the action desired to the system. The animator must not only know what a particular action should look like but also be knowledgable of the processes that make up motion. The animator must understand the capabilities of the character. The animator must be able to describe the desired motion to the system in a language that it understands, as well.

It must be stressed that the goals of character animation and robotics are very different. Robotics is concerned with generating efficient motion. Character animation is concerned with generating expressive motion. The expressive quality of a motion is not algorithmically definable. A character within a goal-directed system can be directed to open a door and it will. But it must be remembered that animated characters develop their personalities through their actions. Therefore, each character will open the door in its own distinct way. The animator will need the control necessary to construct individualistic movement.

Where forward kinematics solves for the end of the limb for given joint angles, inverse kinematics performs the reverse operation of solving for the joint angles for given positions of the base and end of the limb. Inverse kinematics are central to the development of Animation Software for Simulating Articulated Motion (SAM) at the Department of Electrical Engineering at the Ohio State University by A. A. Maciejewski and C. A. Klein. Applying pseudo-inverse matrices to the Jacobian matrix, Maciejewski and Klein provide an elegant mathematical solution for primary and secondary goal control with a minimum amount of calculation. Appendages with more than six degrees of freedom (e.g. possessing more than two joints, 3 axes of rotation each) can reach any primary goal in an infinite number of configurations. Their equations calculate the joint rotations required "to best achieve the desired task under any circumstances" [34] p. 10. The best solution is modifiable through the specification of multiple secondary goals.

> Take for example, the animation of a baseball player at bat. While the trajectory of the hands and the bat are controlled as the primary goal it can be desirable to control the movement of the elbow in order to invoke a particular batting style ... this secondary goal should be satisfied under the constraints imposed by the primary goal [34] p. 13.

Goal-directed motion is the most sophisticated and complex approach to the problem of character animation. Goal-directed systems often make use of research from artificial intelligence and robotics and are complex research areas in themselves. The research in goal-directed animation systems has been productive, but it may be some time before they are flexible and friendly enough to aid the character animator with complex, expressive motion.

7.4. Constraints

Constraints restrict the range within which the system can search for solutions to motion specifications. "For human movement such constraints usually express relationships between objects: contact, guide, support, and surround ... Constraints usually appear as physical limitations on the movement of an articulated object" [2] p. 29. Constraints may exist for joint rotation, balance or collision detection. An animation system is not likely to be based upon constraint control, however, constraint is a powerful control method when used in tandem with any of the other methods (see System Specifications, Multiple Control Methods).

7.5. Simulation

Many movements of an object result from the relationship of the object to its environment. This relationship can often be mathematically described and modeled. A simple example is the relationship of a wheel's rotation to its translation across the ground.

The implementation of a simulation algorithm can take a variety of forms. The programs can be highly interactive or provide for no interaction beyond the specification of the initial parameters. A program may allow the animator to watch a real-time vector playback of the animation while varying parameters interactively. Another program may be run from a non-graphics terminal, producing animation that must be rendered by a display algorithm before it can be seen.

Simulation often offers the ability to specify a motion completely with a few parameters. A projectile motion may need initial values for the force of gravity, velocity, direction, and position. From these few values, an animation sequence can be produced. The refinement of the motion is easily accomplished by changing the necessary parameters and recalculating the animation.

Many of the secondary movements commonly animated within an interpolation system, involving "squash and stretch" and "overlapping action", are designed to give the characters a feeling of weight and solidity. Many such movements are simple exaggerations of the effects of the dynamics of moving bodies. The dynamics of motion are a common simulation model.

Motion that models reality precisely is not always desirable. This important point is rarely acknowledged by the computer animation community. Analytical and simulation controls can model motion precisely but the resulting animation often lacks a sense of "life". The action seems mechanical. The motion may be "too subtle to be readable" [48] p. 36. Two dimensional "cel" animators discovered that it was not enough to render movement realistically. More expressive motion demanded exaggeration beyond the action found in reality. Schweppe found that the movement of her algorithmically generated spring oscillation was not great enough to attract the viewer's attention. This was corrected by exaggerating the motion. Two-dimensional animators often clarified an action by exaggerating the primary action and reducing, or even eliminating, the secondary motions. The animation software must provide the animator with this type of control necessary to generate expressive motion.

The animator has much less direct control over the action of objects within a simulation. Interpolation methods, movement functions, and goal-directed systems all provide the animator with a means of positioning the object at a precise location at a specific time. Using simulation methods, the animator controls object motion indirectly by manipulating the system within which the object resides. The animator can influence the motion by altering parameters.

Using simulation, the animator surrenders his "hands on" control and allows the system to animate a movement as the consequences of a system constructed by the animator. The animator may wish to construct a "real world" model, where objects possess mass and are effected by gravity, inertia and friction. "Simulation systems model the physical effect of concurrent movements in a general environment independently of graphical representations" [2] p. 30.

Blinn uses a simulator that models Kepler's laws and simulates planetary astronomy to produce animated films of spacecraft planetary fly-bys [7] p. 11.

Susan Amkraut, of CGRG, has approached the problem of movement control through the use of attractor and repellant fields. Positive or negative fields, stationary or moving, are used to control the movement of any number of objects. Each object is also aware of the repellant force of every other object and reacts accordingly. Several objects controlled in such a system behave in a manner reminiscent of a flock of birds. Each object moves along its own path seemingly sensitive to the location of all other objects in the formation [1].

Simulation is a common method of motion control. It can benefit the character animator wherever a transformation can be described as a consequence of a system. However, the animator who wishes to use this control method must understand that the animation must work within the laws of the constructed system. An animator trying to animate a character executing an athletic maneuver, within a system that models dynamics, would be in the position of a coach telling an athlete to move an appendage into the proper position at just the right moment. The animator must not only know what the desired action will look like, but understand the physics that define it.

In some cases, the animator may find a strict simulation system inhibiting, preventing a variety of actions that require suspended or even conflicting system laws. Many of the best comedic effects of 2d animation play on the temporary suspension of physics. The "Road-Runner" cartoons are an excellant example of physics gone wild. The broken physical laws do not form a coherent system that can be modeled. Many gags are based upon the simultaneous existance of two mutually exclusive systems of physical laws. A common example involves the contrast of the effects of gravity upon two characters. One character is as completely uneffected by gravity as the other is subject to it (the humor is most effective when the characters are placed hundreds of feet above a canyon floor).

CHAPTER 8

Character Structure

The design of the animation system must consider the structure of the characters it will animate. Schweppe provides a convenient classification system within which to discuss considerations concerning the object types from which the characters can be constructed. Schweppe classifies object types as Plastic, Rigid, Plastic Articulated, Rigid Articulated, and Composite Objects [48] pp. 21-37.

8.1. Rigid objects

Rigid monolithic objects are objects which have no moving parts and never alter their shape. Their size and proportions never change. Their movement is limited to that possible by translation or rotation. Rigid objects are the most common to computer graphics. Viewers are often fascinated by the computer's ability to portray a rigid object in a variety of orientations with proper perspective.

8.2. Plastic objects

Plastic objects consist of a single pliable unit. The method of shape deformation can be performed by scaling the object, through algorithmic deformation or shape interpolation. Schweppe proposes that, "given the proper controls, almost anything can be animated using plastic objects" [48] p. 24.

Schweppe offers real life examples such as an inflating balloon and a bouncing rubber ball. The changing shape of the object can be approximated simply by scaling the object along one or more axes.

More complex functions, such as twisting, bending, and tapering of geometric objects, can be simulated by solid modeling operations [5] p. 21. Algorithmic deformations can model undulating objects such as a snake, a flag as it blows in the wind, and fire [48,31].

Shape interpolation provides the most plastic of the manipulations, transforming one object into another.

Experiences with shape interpolated objects are related above and by Van Baerle [56] pp. 7-11. Shape interpolation requires particular care in the construction of the objects to be transformed. Objects improperly constructed can constrict or pinch during transformation.

A number of object generation and manipulation functions can assist the animator using this technique. Visual, interactive object editors (e.g. Inspect at CGRG [33] pp. 78 - 101), should allow the animator to view the object from a number of viewpoints and interactively identify, select, and move individual points and polygons (if polygonal mesh).

Procedural object generators can ease the difficulty of constructing similar objects with the same description. Objects are constructed by a programed procedure. Usually, variations can be produced through the use of parameters. For example, a program designed to construct a twisted ribbon can construct several ribbons, varying the thickness, length and number of twists of each. Each ribbon will have the same number of points and the same polygon descriptions, but the location of the points may be very different. They should produce clean transformations [53,28,9].

As with key-frame interpolation, many different interpolation schemes can be used to transform the shape. Linear interpolation will cause the most apparent distortion of apparent length. This problem, described by Van Baerle and Kochanek and Bartels is most apparent when a flexible object attempts to simulate a rotation of an appendage [56] pp. 24-26, [29] p. 34. Rather than rotate the endpoint of the appendage about the perceived joint, linear interpolation moves the endpoint in a straight line from one position to the other. As the angle of rotation increases, the error becomes more apparent. Solutions involve the generations of more "key" objects for the motion as described by Van Baerle or using interpolating spline curves as described by Kochanek and Bartels [56] pp. 24-26, [29] p. 34.

It is desirable to have a visually interactive system that allows the animator to check the interpolation of a shape by itself, before it is placed into the context of a scene with other objects. Shapex, in use at CGRG, provides a vector display of the objects to be interpolated and a real-time playback of the interpolation. The animator may also "step through" the animation one frame at a time [21]. This system might also provide for the interactive manipulation of the cubic splines (see Control

Methods, Interpolation) to change the movement and timing of the shape interpolation. Immediate visual playback would aid the review and refinement of the resulting interpolation.

8.3. Rigid articulated figures

8.3.1. hierarchy of links and joints

Rigid articulated figures are constructed of a number of rigid objects that are linked together. Rotational joints between these objects provide a convenient method of controlling the object's movement. A hierarchy of rigid object links and joints provides both a method of organizing the elements into a logical representation for the animator and a way of describing to the system how the rigid links are related and are effected by each element's transformation [48] pp. 26-28. There are two popular methods of manipulating the positioning of links of the object. The first, known as forward kinematics or direct kinematics, requires the rotations or angles of the joints to be specified and the positions of the links are calculated [63] p. 24. The second, inverse kinematics or Jacobian control, requires the specification of the ends of a jointed limb and calculates the necessary rotations or angles of the joints [19,34,45], [63] p. 24.

Whatever method is used to control the links and joints of the structure, the animator may find the reporting of local, object space, and absolute, world space, locations and rotations of joints, links and end-effectors valuable in non-goal-directed control situations where the structure must interact with other objects.

8.3.2. surface generation

The jointed, articulated structure provides a convenient method of manipulating a figure, but the appearence of separate, distinct links is often aesthetically undesirable. The problem is one of generating a smooth "skin" over jointed appendages.

A surface can be generated about a curve defined by the joints. The positions of the joints are used as control points to generate a spline curve of the limb.

Rigid objects can be propagated along the curve to form an object similar in appearance to the flexible banded metal neck common to desk lamps. This scheme has been utilized by Lundin of NYIT (New York Institute of Technology) [32].

A single instance of a rigid object can be "bent" to conform to the curve as performed by Longtin at CCP (Cranston/Csuri Productions, Inc.) [31].

Procedurally generated objects can be constructed about the curve. Tripp's Tog Carlson's DG (Data Generation program) at CGRG build tubes about space paths [53], [9] pp 53-58.

The links can be left untouched, but are joined with smooth fillets which are generated for each frame [30].

Blinn's blobby primitives (see Modeling, Diffuse Models) generate their surface at rendering time at a specific distance from source points. Blobby primitives can be combined into articulated objects and would maintain a smooth surface over dynamic forms [8].

Zeltzer proposes that a "surface knowledge base", a set of rules that specified how the surface near each joint behaved as it moved" could be maintained [63] p. 121.

8.3.3. Generation time

The plastic surfaces within "Snoot and Muttly" were animated by interpolating between key objects. The interpolated objects were temporarily generated at rendering time and immediately removed from the system. Even so, complex motion sequences demanded many key objects. A large amount of data was necessary for the animation of all of the plastic surfaces in the film. The demand for storage space for the many necessary surface objects was exessive and required careful management of data.

The generation of surfaces at rendering time would solve the storage problems. Blinn's blobby molecules are generated at rendering time, but most other systems build surface objects. The nature of surface representation should be a concern of the animation system.

8.4. Plastic Articulated Figures

The Plastic articulated figure is a structure of plastic objects linked together in a fashion similar to the rigid articulated objects, except that,

> the joint descriptions must be plastic as well . . . the joint descriptions must change as the shape of the object changes. If the upper arm elongates the elbow would move further from the shoulder [48] p. 30.

8.5. Composite Articulated Figures

"Composite objects are constructed of both plastic and rigid parts which are linked together" [48] p. 30. Snoot and Muttly are composite objects. Their legs, tails and necks are plastic, possessing plastic joints at the feet and head. The joints for the feet and head had to move as the legs and head changed their shape. The bodies, heads, beaks and feet are rigid. Rigid joints exist in the beak, for the eyes, and at the bases of the neck and tail. The rigid joints did not move in relationship to the base object.

Some objects require the use of both rigid and plastic joints. The neck was jointed to the body with a rigid joint that never moved in relationship to the body. The head was attached at the other end of the neck with a plastic joint that moved as the end of the neck moved.

CHAPTER 9

Modeling Schemes

9.1. Polygonal Mesh

Polygonal mesh are collections of vertices, edges, and polygons. It is assumed that the reader is familiar with this common representation. Polygonal mesh are discussed in detail in [17] pp. 507-510, and [38] pp. 301-307.

9.2. Mathematically described Surfaces

Curved surfaces can be approximated by polygonal structures, but lack smoothness. There are mathematical methods which can model curved surfaces more accurately and smoothly. Surfaces can be described mathematically algebraically or parametrically [6] pp. 6-7.

9.2.1. Algebraic Surfaces

These surfaces, generally treated within algebraic geometry, are described by the points which solve the algebraic equation:

$$F(x,y,z) = 0$$

Algebraic equations are quick and easy to calculate. Their surfaces are compactly described in a single equation [6] pp. 6-7. "Surfaces defined algebraically are inherently well suited to raster conversion algorithms" [8] p. 239.

Algebraic functions of the first order describe planes. Surfaces of the next highest order, quadrics, are generally non-planar and are represented by second degree equations. A sphere is described by the equation:

$$x2+y2+z2 = r2.$$

"Quadric patches, while smooth in appearance, are not suitable for modelling arbitrary forms, since they don't provide enough degrees of freedom to satisfy slope continuity between patches" [11] p. 1. The joints where quadric patches meet are difficult to smooth.

9.2.2. Parametric Surfaces

Parametric surfaces, generally treated within differential geometry, are described by Blinn as,

> those points generated by three bivariate functions as the parameters vary [6] pp. 7:
>
> $$X = X(u,v)$$
> $$Y = Y(u,v)$$
> $$Z = Z(u,v)$$

A bivarient, or bicubic, patch interpolates or approximates surfaces with four bounding curves defined by cubic equations of two parameters. Varying both parameters from 0 to 1 defines all points on a surface patch. The most commonly discussed surface patches are the Hermite, Bezier, and the B-spline [6] pp. 17, [17] pp. 523-536, [38] pp. 313-325, [46] pp. 157-187.

The bicubic patch offers the advantage in that it can be sudivided very quickly, reducing rendering time [11] p. 12.

Because of the complex yet intuitive nature of patch construction, the technique lends itself to a visually interactive surface design system, whereby the animator can locate and move the control points of the surface and observe the resulting changes. Newman and Sproull suggest a number of properties they feel are important for designing curves [38] pp. 311-313.

(1) The use of control points to control the curve's shape in a predictable way.

(2) A curve is not a graph of a single-valued function of a coordinate but is mutliple-valued.

(3) The curve should be axis independent. The shape of the surface must not change if the reference coordinate system is changed. For example, when the control points are rotated 90 degrees, the curve should rotate 90 degrees and preserve its shape.

(4) The designer should have the option of global or local control of the surface. When control points are manipulated, the surface may only be effected in the region surrounding the control point or the entire surface may be changed.

(5) The mathematical representations should possess a variation-diminishing property, tending to smooth the designer's control points.

(6) The curve generation technique should provide the designer with versatility, allowing a wide variety of shapes.

(7) Complex shapes are often modelled by several curves pieced together like a quilt. The designer should be able to control the order of continuity at the joints. Zero-order continuity ensures only that the two curves meet; first-order ensures that the two curves are tangent at the joint; second-order ensures that the curvatures are the same about the joint.

9.3. Particle Systems

"A particle system is a collection of many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into the system, move and change from within the system, and die from the system" [43] p. 92. Particle systems may be used to model such things as fire, water, clouds, grass and trees.

9.4. Density Functions

The primitive is modelled "as a density function at a spatial location... The surface is defined as those points where the density function equals some threshold amount" [8].

A collection of primitives is modeled by summing the contributions of each primitive separately, creating the effect of "blobbiness". As the surfaces of two primitives approach each other, the surfaces are attracted to each other and seamlessly bond. The primitive is described by two parameters, radius and "blobbiness". Complex objects with smooth surfaces can be constructed from primitives. Primitives can be any general quadric, including spheres, ellipsoids, cylinders, and planes.

9.5. Volume Modeling

Complex objects are described as simple primitives, cubes, cylinders, wedges, or spheres, related by the Boolean operators AND, OR, and NOT. Objects can typically be described as the union of two primitives, their intersection or the subtraction of one from the other [9,62].

CHAPTER 10

Bookkeeping

The management necessary for the large amount of data involved soon becomes an apparent problem in the production of character animation. There is likely to be a large number of programs, object files and animation and rendering scripts that must be organized so that they can be quickly found. The progress of the animation process must be monitored for each character of every scene from the animation stage to the post-processing stage. A large portion of the animator's time can be spent oragnizing and searching for this data. A bookkeeping system could relieve the animator of much of this burden.

The bookkeeping system would operate at two levels: one level manages the animation system itself; the other manages information particular to each animation project.

10.1. Track Elements for Each Animation

10.1.1. Data

The system would organize the data required for each character or prop. These include: constructive elements, levels of detail, and simple representations designed to represent non-polygonal data on vector display.

10.1.2. Animation Process.

The animation scripts maintained by the animation system can be conveniently organized by scenes, shots, and character. In some cases, when a character's action spans several shots or scenes, it may make more sense to maintain the script in its entirety, to ensure continuity. For this reason, it may make more sense to organize scripts by scene and shot within character. The scene and shot numbers should be consistant for all characters.

The computer allows an animator to save a script in its current state before a risky change is attempted. If modifications are not successful, the animator can return to the prior copy of the script and try again. Unfortunately, this may create a number of versions of the same script. The bookkeeping system needs to organize versions of a script and be able to provide the animator with the version desired.

The bookkeeping system should be able to provide the animator with a list of all objects required by the animation system for a particular script. Better yet, it would check to see if the objects were currently available and only notify the animator of those not found.

10.1.3. Rendering Process

Once a scene is designed, the animation system may provide the animator with a script for the rendering software. The bookkeeping system must organize these scripts.

After final calculation, abbreviated scripts may be produced by an animator to correct and recalculate faulty frames. These abbreviated scripts need to be paired with the parent rendering script for storage.

If rendering calculations are being performed on more than one machine, the system should note the machine where each particular script or frames are calculated.

The system should also check that all objects required by a particular script exist and are available to the rendering program on the machine to be calculated on.

10.1.4. Cels and Backgrounds

Animation cels should be tracked by scene and character. All cels should possess a frame number that is consistant for all cels of a scene. A completed image is created by merging all cels of the same frame number.

The system should track movement of cels from the machine upon which they were calculated until they are placed onto tape and always should know location of any given cel.

10.1.5. Finished Frames

Finished animation frames should be organized by scene.

Any dissolves, fades, any post-processing necessary should be noted by scene.

The order in which "cels" are to be "layered" for scene should also be recorded. Ordering the merges is unnecessary if pixel depth information preserved with the cel, see System Specifications, Rendering Considerations, below.

10.1.6. Animator Log

The bookkeeping system should also provide a diary function for the animator. Logging entries should be quick and easy, available from any level of the system with a single command. The logged note should be appended to the end of the log file and tagged with the date and time.

The animator could use the diary to log problems and make notes on animation progress, animation techniques or ideas.

10.2. System Organization

The system will most likely comprise a number of smaller programs.

In discussing the experience garnered from the production of several computer animations Jim Blinn states that,

> The first conclusion is that you cannot expect to develop one program that does everything. It is much better to have a collection of smaller programs, each of which is relevant to some small portion of the whole process. This allows the sections of the process to be thought about and debugged individually without needing to consider all aspects of the system at once [7] p. 1.

Clearly, there must be some means of controlling this software.

The system should assist the animator searching for a particular program. It can do this by cataloguing programs by name, function, and keyword.

Simple naming conventions for each classification help the user to deduce a program's likely location. The system could: query the user installing the program for the classification information; search a library of standard abbreviations and naming conventions; and log the appropriate catalogue entries. An automated system could promote documentation of programs installed in the system.

For the user searching for a particular program, The system should accept as many clues as the user can give and respond with a list of programs that fit the desciption. The name search should allow the user to give portions of a name, and also search for all names that are similar if an exact match is not found. For instance, the user specifies "mk_object", and the system, unable to find a program by that name, does find a program named "make_object".

10.3. Common Data Area.

In a production environment, a large number of characters and props may be developed. Many of these characters or props may be used regularly in animation. Others may be created for a specific purpose but are placed in storage for possible use again in the future. An animator beginning an animation may wish to check the data library for objects that may suit the purpose as is or with a minimum of modification.

To make such a system work, requires a carefully monitored cataloging system. The system should provide images of requested data immediately to allow an animator to search for an object quickly.

CHAPTER 11

System Specifications

11.1. Animate Characters Individually or in Combination

The animation system should be organized in such a way as to allow the animated characters to be animated individually or in combination with any other characters or objects in the scene. The animation of each character should be completely independent from the animation of every other character or object. This allows several animators to work simultaneously on individual characters in a scene without the inconvenience of waiting their turn to work on a singular script for all the characters. It is likely that a great deal of the screen time will be devoted to character interaction. However, a great deal of setup and motion debugging for a single character can be done with little or none of the scene present. It would naturally speed up the animation process if each animator working on a scene could have access to the individual characters concurrently.

When the story demands interaction between any combination of characters and objects in the scene, the system should be able to combine the necessary characters and objects temporarily. All characters and objects present in the system would be subject to the control of the animator. With several characters within the system, it would be logical to provide input devices to allow two or three animators to simultaneously manipulate their respective characters interactively.

Once the characters' interaction is adjusted, the characters can then be worked on separately again for motion refinement. Each character can be calculated separately if the display algorithm maintains the coverage of the pixels (Coverage and the production of 'cels' will be discussed below in Rendering Considerations).

At the very least, the system ought to provide the facility of turning undesired elements "off" [23]. The system will not display these items and the speed of interaction will increase. The production of "cels" for each character is also facilitated.

11.2. Solid Modeling Scheme Independence

Each data type has its own characteristics. Certain data types are more appropriate for objects required to perform certain actions or to have a certain "look". It can be assumed that the animator will wish to animate objects of several different data types within one scene, possibly within the same object. This requires that the animation system be modeling scheme independent.

Blinn describes a system that only deals with a schematic representation of an object, no matter what its type. The representation is constructed from the solid model file by a pre-processor or by hand [7]. Lundin, at NYIT, also describes vector representations constructed for non-polygonal data structures [32].

11.3. Rendering Considerations

The display algorithms and rendering programs utilized in the system will have a direct effect upon the animation possible with the system. Ten basic display algorithms are discussed in "A Characterization of Ten Hidden-Surface Algorithms" [51].

11.3.1. Cel Capability

A major consideration in the construction of a rendering system is whether or not the system will provide the animator with a method of constructing and combining images in a manner analogous to the use of the traditional 2d animation "cel". Using such a technique, the animator is able to calculate and store several elements of the animation separately. When all elements are calculated satisfactorily, they are layered to produce the final animation images.

Porter and Duff compare the construction of an image with the experience of constructing computer systems. According to Porter and Duff,

> Reliance on a single program for rendering an entire scene is a poor strategy for minimizing the cost of small modeling errors. Experience has taught us to break down large bodies of source code into separate modules in order to save compilation time. An error in one routine forces only the recompilation of its module and the relatively quick reloading of the entire program. Similarly, small errors in coloration or design in one object should not force the

> "recompilation" of an entire image... Separating the image into "elements" which can be independently rendered saves enormous time [42] p. 253.

An animated scene is typically composed of several elements, the major characters, environmental objects and a background. Each of these elements can be calculated separately as images analagous to the "cels" of conventional 2-d animation. Each pixel of a "cel" can be assigned an opacity value (coverage). This value is a function of the object's transparency and the percentage of the pixel that the object obstructs. A pixel at the center of an opaque object will have a maximum opaque value, a pixel outside of the object will have a minimum, or zero opaque value, and the value of a pixel on the edge of the object will be in proportion to the objects's coverage of the pixel.

The images can be "layered". The layering process merges two images, giving one or the other precedence, and anti-aliases the edges. The order of the merging is not associative and must be performed back-to-front or front-to-back, depending upon which layer takes precedence in the buffer. Each layer is merged with the composite of all previous layers. Preservation of calculated depth for each pixel or assignment of depth to a layer would allow merging with order independence [57] p. 256, [42] p. 259.

If there is no camera movement in a scene, the environment can be calculated once for a scene. The resulting image is saved and composited with the animated characters and objects for the final frames.

11.3.2. Multiple Data Structures

The rendering system should be able to handle several data structure types in combination. Data structures possess their own particular characteristics. The animator must understand these characteristics when choosing a modeling scheme for a particular object. To produce an animation, an animator may wish to model objects with several different data structures.

As a worst case, the animator should be able to calculate objects of differing data structures as separate images and combine the images to form a final composite image.

11.3.3. Larger than Screen Size Images

Backgrounds larger than screen size could be calculated and scrolled past the screen in a manner analogous to the background paintings in traditional 2d animation [39]. Larger than screen size images could be calculated as "cels" and combined and manipulated individually in a 2 and 1/2 dimensional manner. This approach is unlikely to produce environments that will be mistaken for 3d, but could be used for aesthetic and time conservation measures.

11.3.4. Special Purpose Renderers

Simple, geometric objects can be rendered quickly with special purposed renderers that take advantage of their specific purpose. Samples of such objects include spheres, lozenges, and simple rectangular planes. These objects may have no model in the system beyond that of a simple point to provide the renderer with placement, rotation and scaling information. Additional information may be demanded, in respect to the object generated, such as dimension or translucency. The renderer reads the transformation, viewing and lighting information from the scene description and generates a shaded, 3d image in the buffer, overlaying its current contents.

Blinn uses several special purpose renderers for his spaceflight simulation animations. Blinn's simple routines produce anti-aliased dots for a starfield, or a 2-d blurred spot for the sun. Other, more complex renderers read the viewing and lighting information and draw textured spheres for planets and textured translucent rings [7] p. 13.

11.4. Multiple Control Methods

Each of the motion control methods discussed above is particularly suited to a class of motion. It is rare when any of the techniques discussed are used in isolation. It is likely that as the animator becomes familiar with the system, the animator will begin to conceive of complex movements that will require more than one type of motion control for elegance of execution. Ideally, all control methods would be available in series or parallel operation on any object. In the worst case, the animator should be able to playback animation and view objects controlled by a variety of control methods simultaneously. Several examples of limited interaction of control methods in animation systems follow.

11.4.1. Movement Functions within an Interpolation System

Twixt, discussed earlier, is an event-driven animation system. Based upon an interpolation control method, it handles parameter values as positions upon interpolated paths like a movement function.

Interpolation schemes as described above (see Controls) perform the interpolation of key values. The key values have an established positioned in time relative to a frame number. If a parameter is unspecified for a given frame it is calculated by interpolating its value from the neighboring key values.

The track can be abstracted to the concept of a path which graphs a parameter's value over time. This concept is not new, Two-dimensional animators would "chart" character action as a path on an exposure sheet [52] pp. 290-292. Computer animation allows the animator to specify key values as control points for generating the path. A variety of interpolation schemes are available for this purpose. Each interpolation scheme imposes its own characteristics upon the resulting motion.

Linear interpolation produces a linear change between key values. Linear interpolation of position values results in straight-line movement between key locations. The location of key values can be seen in the resulting motion as sudden changes in direction or speed. The animator must construct his key values carefully to alleviate jerky motion.

The application of "eases" can smooth abrupt velocity changes. However, the path of movement remains linear.

Spline-curve interpolation techniques produce smoother, more natural curved movement similar to the 2d animator's concept of "arcs". Two-dimensional animators made the observation that very few characters moved in a mechanical, linear fashion. Most movement is more accurately described by curves [52] pp. 62-63. Smooth parameter paths, or graphs, can be produced from a few key values acting as control points for cubic spline interpolation.

A Catmull-Rom scheme requires a minimum of four control points, or values, for a parameter. The resulting parameter curve will pass through all of the specified values.

The Bezier method requires a minimum of four control points. The parameter curve will pass through only the first and last points. The remaining points are used to influence the curve, but the curve will not pass through them.

The B-spline technique requires a minimum of four control points. All four control points influence the curve, but the resulting curve will pass through none of the control points.

Cubic spline curves are discussed in more detail by Catmull, Rogers and Adams, Foley and Van Dam, and Newman and Sproull [11] pp. 53-61, [46] pp. 139-155, [17] pp. 514-523, [38] pp. 315-324.

The system can apply different interpolation schemes to different portions of the same track. This is known as "patching" [24]. There may be a jump at the joints. This is undesirable when the application of cubic splines was used to smooth the parameter curve.

Track manipulation may be difficult to perform if it is only reflected within the animation of the interpolation. The graphic representation of the parameter paths would provide the animator with a visual model of each track. The animator can easily see jumps between "patches" as discontinuities. The motion can be smoothed by interactively manipulating the control points to force continuity at the joints [22]. For cubic interpolating splines, the character of the interpolation can be further controlled visually. The visually interactive system described by Kochanek and Bartels allows the animator to control what they refer to as a spline's local tension, continuity, and bias to modify the interpolation of key-frames [29].

A disadvantage of the "key-framing" system is the fact that specific interpolation schemes require a certain number of key-frames to function as discussed above. This is a mathematical limitation and it requires that the animator be aware of the number of key values required by the chosen types of interpolation. Key events must be choosen accordingly. This may have an effect upon the quality of the motion. Badler feels that,

> this tends to restrict his control of the actual flow of the movement, so interactive refinement of a specific sequence is essential [2] p. 28.

A visual interactive system for viewing tracks as graphs of interpolated parameters could restore an animator's sense of the movement flow.

The abstraction of "tracks" allows the animator to replicate a section of a track to create "motion cycles", or copy the track for another object to create synchronized motion [18] p. 181. Two tracks can be related by a function. For instance, one track could be the inverse of the other across one of the axes, producing a mirror image [24].

Interpolation systems typically require the animator to position each segment of each appendage separately, working from the base of the appendage to the end in a manner similar to forward kinematics. An inverse kinematics function within a interpolation system would allow an animator to construct the key positions by visually positioning the ends of the appendages in their desired locations in 3d space. The system would calculate the necessary joint angles to bend the appendages into the desired position.

If at any time the animator were not satisfied with the resulting pose, the rotations of any of the joints could be adjusted interactively. Perhaps the end of the appendage could be "locked" at a specific point to allow the animator to vary a joint and have the remaining joints recalculated to maintain the appendage's end position.

11.4.2. Constraint Techniques within a Interpolation System

An animator may request collision detection and control, constraint techniques, within a key-framing system. The system would alert the animator to any condition in which a character intersected with any object in the scene. This feature may even go so far as to prevent the animator from positioning characters in positions that would cause intersection between the character and any other object (e.g. the floor or another character). In an interactive system, the characters would take on the quality of a solid. The animator would be unable to place their feet through the floor, or walk them through a wall.

11.4.3. Interpolation Within a Goal-Directed Control System

Within a goal-directed environment, an animator may simply wish for the ability to interactively adjust the poses produced. The animator may also want to animate the body of a figure with a goal-directed system, but animate the facial features with an interpolation system.

11.4.4. Goal-Directed and Simulated Motion Controled by Interpolation System

Richard Lundin of the New York Institute of Technology, uses a variety of control methods to construct moving vehicles in three-dimensions. A visually interactive interpolation system, BBOP [60], is used to select key positions of vehicles. The positions are expressed as parameters in within an animation data base. The selected vehicle positions function as control points for the generation of paths which control the motion of the vehicles in the manner of a movement function. Simulation programs are used as post-processes to calculate transformations for the leg, wheel or track motion necessary to support the vehicle along the specified path over uneven terrain. The calculated transformations are written back into the data base where they can be modified with BBOP.

Steering and wheel rotation are calculated from the distance traveled and the arc of a path segment. Simulation programs ensure that legs and wheels will contact the non-planar surface. Parameters for stride length, heel strike angle, toe-off angle, and timings for step events are provided to the program to define walk characteristics. "The walk program is then invoked to create the motion of the legs from purely geometrical considerations such that the supporting feet contact the ground and swinging legs clear the ground" [32] p. 5. These programs can be intrepreted as goal-directed control in that the goal of the leg or wheel movement for a single frame is determined by the vehicle's position as given by the path and the description of the ground surface. The supporting leg or wheel must contact the surface and the required joint angles are solved for geometrically. Leg positions between key-positions is interpolated.

A simple dynamics model was developed "based on damped mass-spring systems [35] p. 336" [32] p. 7. This simulation program models the bounces of vehicles as they move over uneven terrain, their pitch due to acceleration and their roll as they turn. Ballistic equations are utilized to model trajectories of thrown or falling objects. In the case of a thrown object, the initial position and velocity are derived from the positions of the throwing hand in the BBOP data base.

11.4.5. Goal-Directed Motion Controled by Movement Functions and Augmented by Simulation

Michael Girard and A. A. Maciejewski, of the Computer Graphics Research Group at the Ohio State University, are developing the goal-directed, Figure Animation System (FAS). Their initial efforts have been focused upon "developing a general model for legged locomotion, in contrast to systems geared toward the simulation of human motion" [19] p. 2. They describe an efficient implementation of Inverse Jacobian control to solve the inverse kinematics problem inherent in goal-directed motion. It determines the joint angles necessary to place a foot at a desired location and to maintain its position in world space as the body moves over it.

A step is described by the desired path of the feet, and the preferred angles of the joints and their gain (priority to be satisfied) at "key leg positions" as in an interpolation scheme. The primary goal involves positioning the foot along the catmull-rom spline from one foothold to the other. The desired joint angles are handled as secondary goals (see Control Methods, Goal-Directed Motion). The parameters can be instanced for as many legs as are desired.

The animator provides the figure's path as a horizontal cubic spline. The position and velocity of the figure along the path are determined by arclength calculations, given the desired speed. FAS generates the leg movement of the desired gait to support an object moving at a given speed along a given path. The actual path of the object is maintained as close as possible for the desired gait and velocity.

Limited dynamics are provided to modify the position of the body in response to the simulated upward forces of the legs and downward force of gravity. This provides a body motion that relates naturally with the movement of the legs.

11.5. Planned System Growth

It is unrealistic to expect that a system design encompass every possible application. It is almost guaranteed that special purpose programs will need to be continually generated as animators tackle larger and more complex problems. The animation system should be designed not only to allow for this growth but also to encourage it. The system should provide a library of common programming and animation functions to facilitate the quick generation and integration of special purpose programs.

11.6. Object Substitution.

The system ought to provide a simple means of substituting one object for another. In the absence of a representation system such as the one described above (Solid Modeling Scheme Independence), the substitution of one object for another would permit simplified "stand-in" objects to be used in the motion study stage. The response time of the animation system could be positively affected. More complex objects would be substituted for the "stand-in" for the calculation of the final images.

11.7. Interface

There are a great variety of input devices available to the animator, the keyboard, joysticks, track balls, puck, light pen, knobs, etc.

Choosing the proper device depends upon the particular application, or motion control method. An animated helicopter flight may at first seem to lend itself most appropriately to a joystick controlled simulation system. But an animator who wishes to fly a specific path may prefer to input space coordinates numerically through the keyboard and generate a space path for the copter to follow algorithmically. An animator more concerned with the helicopter's movement on the screen may wish to use a interpolation system for motion control, and use a screen space system as described above, locating key locations and sizes on the screen with a light pen, to describe key positions.

The key is to offer the animator choices. Animator's generally have their own preferences. Some prefer to enter numeric data and verbal commands through the keyboard; others prefer non-verbal devices.

11.8. Human Readable/Editable Script

Some systems output and store animation sequences in the form of a human readable/editable script [23,32]. The script can provide a valuable interface to the user. A readable/writable script can act as a "back door" the animator can use to circumvent any deficiencies in the system. Functions not accounted for by the system can either be performed upon the script by hand or algorithmically. Functions that are slow may be avoided by utilizing a quicker method to manipulate the script directly. Such a script releases animator from unforseen limitations of the system.

Animators at Robert Able & Associates would edit the motion script to do the following [54] p. 5:

(1) directly and explicitly enter objects and simple transformations into script.

(2) Shift a motion to a different point in time.

(3) Reverse a motion sequence.

(4) Create a repeating motion cycle.

(5) Copy motion sequences.

The animator can sometimes take shortcuts around a system by editing desired changes into the script. The animator may find it quicker and simpler to edit a change in a script than to make the change through the animation program. The animator can sometimes check animation in the script without using a graphics terminal to review it.

An editable/writable script ultimately provides a form of device independence. The user does not require a graphics terminal in order to use the system.

11.9. Device independence.

Animation systems can be designed as single-user systems dependent upon limited graphic and input devices. There may be only one workstation available within an environment. Some of the available time on the system will be needed by the software development staff, and the remaining access must be divided among the animators. When any of the graphic or input devices are disfunctional, the animators may find themselves unable to work.

With a verbal command language, it is not necessary to have access to a graphics terminal to gain access to the animation system. Access to the system would be available from any terminal. A verbal command language would allow the animator to construct a rough script, check an animation or make minor changes without entering the animation system. Information International, Inc.

using the animation/graphic language developed for ASAS, could outline a script for an animation before any of the objects were available [44] p. 295. The animator, prepared in this way, can make better use of the available time on the graphics and input devices.

11.10. Frame Independence

In conventional animation techniques, animation frames are produced to satisfy the 24 frames/second projection rate of film. Twenty-four frames of animation are produced for each second of animation. Computer-generated animation can also be recorded directly on videotape. Videotape has a frame/second ratio of 30 frames per second. These are the frame/second ratios in the United States. Other countries may use a ratio of 25 frames/second for both video and film.

An animation system can follow the convention of traditional animation and link an "event" with a frame number. However, if an animator were to switch the final format from film to video midway through the production, the frame numbers must be changed to maintain the same timing. A conversion is possible by mutliplying all frame numbers by the desired factor.

Computer animation offers the animator the possibility of working with absolute time without regard to the frame/second ratio demanded by the final format. "Events" can be tagged in terms of seconds. The necessary number of frames per second can be determined and generated at the time a scene description for rendering is produced.

The conversion between time and frames is subject to temporal aliasing. The frames generated by the frame/seconds conversion function may not capture the extremes. An algorithmically controlled spring, oscillating in synchrony with the frames, appears to be motionless in the animation. A ball dropped and bouncing across the floor under simulation control may never be caught striking the floor in the animation frames. In one frame it is approaching the floor, in the next frame it is rising above it, the actual strike occurring between the frames. The missing strike, however fleeting it is, removes the "punch" from the animation.

The system should be intelligent enough to ensure that key positions do not mathematically fall between frame numbers but are shifted to the nearest frame.

using the animation/graphic language developed for ASAS, could outline a script for an animation before any of the objects were available [44] p. 295. The animator, prepared in this way, can make better use of the available time on the graphics and input devices.

11.10. Frame Independence

In conventional animation techniques, animation frames are produced to satisfy the 24 frames/second projection rate of film. Twenty-four frames of animation are produced for each second of animation. Computer-generated animation can also be recorded directly on videotape. Videotape has a frame/second ratio of 30 frames per second. These are the frame/second ratios in the United States. Other countries may use a ratio of 25 frames/second for both video and film.

An animation system can follow the convention of traditional animation and link an "event" with a frame number. However, if an animator were to switch the final format from film to video midway through the production, the frame numbers must be changed to maintain the same timing. A conversion is possible by mutliplying all frame numbers by the desired factor.

Computer animation offers the animator the possibility of working with absolute time without regard to the frame/second ratio demanded by the final format. "Events" can be tagged in terms of seconds. The necessary number of frames per second can be determined and generated at the time a scene description for rendering is produced.

The conversion between time and frames is subject to temporal aliasing. The frames generated by the frame/seconds conversion function may not capture the extremes. An algorithmically controlled spring, oscillating in synchrony with the frames, appears to be motionless in the animation. A ball dropped and bouncing across the floor under simulation control may never be caught striking the floor in the animation frames. In one frame it is approaching the floor, in the next frame it is rising above it, the actual strike occurring between the frames. The missing strike, however fleeting it is, removes the "punch" from the animation.

The system should be intelligent enough to ensure that key positions do not mathematically fall between frame numbers but are shifted to the nearest frame.

11.11. Temporal Aliasing

Computer-generated animation are typically rendered as descrete images of environments as they are at specific moments in time. It is impossible to identify moving objects from a single frame. All objects are rendered equally motionless. In film, moving objects leave evidence of their motion. Each frame records the environment over a period of time. That film is subject to temporal aliasing is evidenced by wagon wheels that appear to rotate backwards, Yet computer-generated animation suffers additional problems.

> The most common artifact of temporal aliasing may be strobing. A rule of thumb is that strobing becomes serious in an unblurred image which is moved one one-twenty-fifth of a screen width or greater; the problem is increased with sharp edges and high contrast . . . a soft background reduces the problem [61].

If the final format will be video, another solution amounts to projecting twice as many images per second. Video frames are generally displayed at a rate of 30 per second, while the screen is refreshed at a rate of 60 times a second. The first pass for each frame draws the image on the even scan lines, the second pass draws the image on the odd scan lines. The result is two drawings of each frame of alternating scan lines.

The animation frames are calculated to be projected at a rate of 60 frames per second. Only the even numbered scan lines are calculated for the even numbered frames, the odd numbered scan lines for the odd numbered frames. Each pair of even and odd frames is composited into one frame. The resulting frames are recorded onto videotape. When these frames are projected at a rate of 30 per second, the effect is that of a higher rate of display, allieviating some of the temporal aliasing.

The most accurate solution involves the calculation of the motion blur. The most convincing examples of motion blur are those described in Cook, Porter, and Carpenter's "Distributed Ray Tracing" [12]. Ray-tracing is currently too expensive a technique for most computer animation laboratories and their computer animation productions [17] pp.585-588.

11.12. Velocity control

The application of velocity control falls within the realm of algorithmic control, but two methods offer basic, easy to use approaches that deserve special notation: ease-in/ease-out and speed curves.

The concept of "ease-in" and "ease-out", or "slow in" and "slow out", was established in 2d animation [52] p. 62. "Eases" provided smooth motion by slowly easing into or easing out of a movement. They can provide the 3d animator smooth transitions between differing velocities, as well. All forms of parameter manipulation or interpolation should provide easy to use "ease-in" and "ease-out" functions. These should include "ease-in", linear, "ease-out" combinations, commonly refered to as "combination moves". The animator should be free to specify the transition points and beginning and ending velocities, where appropriate [54] p. 3.

Curves can be used to control the velocity of an object as it travels along a separate curve for motion. It is common practice to rely upon the points that specify the object's path of travel to also control the object's speed. The object advances along the path, one point per frame. This is a very inconvenient method as it requires the animator to make changes to the object's path to alter the object's speed. It is also common that the points necessary to define the path and the speed may conflict.

It is more convenient to separate the speed and travel specifications. The object's path is defined as a 3d space curve. The velocity can be determined numerically. However, a visual graph of distance versus time as a 2d "speed curve" can be very useful. The speed curve can be interactively designed, manipulating a few control points to produce a spline curve.

The motion along the path then becomes a matter of moving a specific distance for a specific frame, determined by the "speed curve", rather than advancing a uniform number of disparately spaced points. Distances along a curve can be measured accurately with arclength calculations [25,49].

11.13. Screen Space Composition Control.

Although one of the strong points of computer animation is its 3d orientation, the final image is 2d. And in some cases an animator may be more concerned with the organization of the final 2d image than with the actual placement of objects in 3d space. The primary objective then becomes a matter of organizing the elements in three dimensions to achieve the desired 2d composition.

Animators at Robert Abel & Associates have the choice of describing motion in terms of absolute 3d spatial position or 2d screen coordinates. "Screen coordinates are often very useful when trying to build motions without a direct graphic feedback" [54] p. 1. Screen coordinates are described in terms of field chart positions. "If an apparent object size is specified in addition to its screen location an actual 3D position can be computed based on a particular lens value" [54] p. 2.

11.14. Post-Processing Functions

A system analogous to the traditional animation stand, capable of translating and varying the intensity of the color of individual layers ("cels") and merging them is described by Wallace. A "cel's" horizontal and vertical placement within the picture plane (screen space) can be changed. This cabability could be used to "pan" across oversized backgrounds or foregrounds (see Rendering Considerations, Larger than Screen Size Images).

The red, green, blue and coverage components of the pixels can be algoritmically manipulated to brighten, darken, tint, or vary the opacity of a "cel".

Various post-processing manipulation functions are possible for special effects, blurring images, reversing images, rotating images, alter image dimensions, display images through masks, etc.

11.15. Selective Image Generation

The animation system can produce upon request a scene description of any specified frame or sequence of frames. A sequence can be incremented at any desired rate. The frames can be rendered at any desired resolution level. These frames may be requested by the animator to check sample frames of the animation prior to final calculation or to produce frames to replace faulty calculated frames.

11.16. Low-Resolution, Real-Time, Raster Image Playback

One method of providing real-time, raster playback without recording the sequence on film or videotape requires that the rendering program render images upon consecutive subdivisions of the display screen. The frame buffer is divided into subpicture elements. Each frame of the test is rendered in a successive subpicture. Real-time playback can then be produced by selectively zooming into these consecutive subdivisions at a controlled rate [20].

11.17. The Cyclical Design Process

The production of character animation has always been a cyclical process of review and refinement [52]. Computer animation is no different. It is critical that the animation system be designed with this fact in mind. Robert Abel & Associates have found that,

> the best designs are achieved through a method of continual refinement and testing over the course of a project. The process is characterized by a cycle of design, evaluation and re-design. The objective for the software has been to reduce the length of this cycle" [54] p. 1.

The animation system should support the design cycle, providing swift playback and easy alterations to an animation sequence. A sample design cycle is described:

(1) The animator constructs an animation sequence using the control methods described above to manipulate a character's parameters.

(2) The system provides visual feedback as parameters are set. If a parameter does not produce the desired position, return to (1).

(3) The system produces a real-time vector playback for review of the motion of the sequence. Depending upon the software, system hardware and the data to be displayed, the system may be able to provide real-time calculation. At the very least, the system ought to provide real-time playback of previously calculated vector images. Vector images calculate much quicker than raster images and provide an animator with sufficient information for preliminary debugging of most motion sequences. If alterations are necessary, return to (1).

(4) Major motion problems should be debugged by this step, and the animator is concerned with color and intersection problems. Some motion problems will not be discernable in a vector playback and will be apparent when the character is rendered as solid and shaded. A real-time playback of low-resolution raster images can usually provide the information needed without calculating the animation in full-resolution. Real-time calculation of raster images would be desirable, but the system must at least provide real-time playback. It is rarely necessary for the system to calculate and playback every frame of the sequence. For debugging, it is often sufficient to playback only every other frame, or in some cases as few as every sixth. Elimination of frames speeds up calculation time, as does the elimination of unecessary elements of the scene. Both processes should be performed by the system at the request of the animator [23].

(5) If the animation sequence has been approved in the previous stages, the animator may wish to calculate some selected frames at full resolution. The animator may wish to request specific frames of the sequence or may wish to choose the frames algoritmically, for example, requesting every tenth frame. At this stage, the animator may be checking for minor problems of intersection or the placement of small objects. These may not be visible at low-resolution.

(6) If the full-resolution frames look good, the animation is then calculated in full resolution for recording on film or videotape. This stage is the final checkpoint. Problems may still become apparent at this stage, requiring a single frame or possibly an entire sequence to be returned to (1).

To support such a cycle, the implimentations of the control methods should provide for easy alterations. As an example, we will examine what kind of alterations can be useful within the interpolation control method.

The most frequent type of change will involve the refinement of motion by changing the values of the parameters that control that motion. The timing within an action can be changed by moving the occurance of individual parameters. Gomez calls this process "sliding an event" [23].

The animator may wish to increase or decrease the speed with which an action is performed. The system can mathematically compress or expand a sequence of motion while maintaining relative relationships by multiplying the keyframe numbers or seconds by a common factor [23], [54] p. 4.

The animator will often wish to insert or delete an action from a previously designed sequence. When this is done, the displaced action should be managed by the system. If an action is inserted, the action that previously occupied that position in time and all action following is moved to a later position in time. If some action is deleted, the remaining action is moved forward to occupy the vacated space in time.

11.18. Future Challenges

Current animation system research is primarily addressing the problems of controlling the motion of the articulated figure. Other problems involving character animation remain to be solved. Problems such as modeling clothing and musculature. Both problems require the ability to conform a flexible object about other objects while maintaining its volume. Clothing involves the problems of folding and bending an object while maintaining its original form. The modeling of muscles requires the ability to model pliable objects which change their form but maintain their volume as they are flexed or relaxed and are moved over and against one another.

Current efforts on the problems of modeling the face and its expressions are documented by Frederick Parke [40] and S. Platt and Norman Badler [41]. Brian Gunther, at CGRG, suggests that a system could generate animated facial features synchronized to pre-recorded speech [26].

The solutions to these problems will demand calculations of tremendous complexity by today's standards. But increasing computer power and storage capabilities will make these functions more realistic in the future.

11.19. Conclusion

The challenge is to build an animation system that provides for a growing assortment of movement functions. The functions must be easy for the animator to use to design many types of motion for many types of figures. The system must make it easy to animate several interacting characters. The system should be designed to promote the refinement of animation: quick visual feedback and easy alteration of the motion.

It is difficult, if not impossible, for the animator to outline in advance the functions necessary for a static, self-contained animation program. The system must grow to respond to the changing demands of a maturing animator as well as a maturing audience. It is important for the developers to realize that an animator's needs are constantly changing.

While it is important that the system grow, the core of the system must remain stable. Changes within the system must be controlled. The constant changing of system capabilities, while increasing the potential of the system, may have a detrimental effect upon the animators' confidence in the system.

In the production of "Snoot & Muttly", we estimated that the portion of time devoted to actual animation was between 10 and 20 percent [56] p. 53. That was the amount of time spent designing the animation and describing it to the animation system. The bulk of the animator's time was involved in supportive tasks (e.g. the management of data and calculated images, software development and debugging, processing the animation scripts through the many small, separate, special purpose programs and recording the final animation). I have no doubts that an animation system can handle most of these processes, and free the animator to concentrate on the issues of expressive character animation.

Like most advances in technology, the automation of the supporting processes has the potential to increase both the quantity and the quality of the work. Animators will have a larger proportion of their time to devote to the actual animation of characters. The increased animation activity can be spent exploring and refining expressive qualities of 3d graphics animation. Animators would be free to further investigate the aesthetics of color, texture, movement and spatial organization.

List of References

1. Amkraut, Susan. 1984. Personal communication.

2. Badler, Norman I. and Smoliar, Stephen W.. "Digital Representation of Human Movement". Course Notes, Seminar on Three-Dimensional Computer Animation (July, 1982). ACM SIGGRAPH 82.

3. Badler, Norman I., O'Rourke, Joseph, and Kaufman, Bruce. "Special Problems in Human Movement Simulation". Course Notes, Seminar on Three-Dimensional Computer Animation (July, 1982). ACM SIGGRAPH 82.

4. Baecker, R.. "Picture-Driven Animation". Proceedings, AFIPS 1969 Sprint Joint Computer Conference 34, , pp. 273-288.

5. Bar, Alan H.. "Global and Local Deformations of Solid Primitives". Siggraph '84 Conference Proceedings (July, 1984). ACM SIGGRAPH '84.

6. Blinn, James F.. "Computer Display of Curved Surfaces". Published Ph.D. Dissertation. University of Utah, December, 1978.

7. Blinn, James F.. "Systems Aspects of Computer Image Synthesis and Computer Animation" in Three-Dimensional Computer Animation Tutorial Notes. SIGGRAPH-ACM, 1982.

8. Blinn, James F.. "A Generalization of Algebraic Surface Drawing". ACM Transactions on Graphics 1, 3 (July, 1982).

9. Carlson, Wayne. "Techniques for the Generation of Three Dimensional Data For Use in Complex Image synthesis". Unpublished Ph.D. Dissertation. Computer Graphics Research Group, The Ohio State University, September, 1982.

10. Catmull, Edwin. "A System for Computer Generated Movies". Proceedings, 1972 ACM Annual Conference 1, (1972), pp. 422-431. ACM.

11. Catmull, Edwin. "A Subdivision Algorithm for Computer Display of Curved Surfaces". Report prepared for the Advanced Research Projects Agency. Utah University, Springfield, VA., December, 1974. Distributed by the National Technical Information Service

12. Cook, Robert L., Porter, Thomas, and Carpenter, Loren. "Distributed Ray Tracing". SIGGRAPH '84 Conference Proceedings (July, 1984). ACM SIGGRAPH '84.

13. Crow, Frank C.. "A More Flexible Image Generation Environment". Siggraph '82 Conference Proceedings (July, 1982). ACM SIGGRAPH '82.

14. Csuri, Charles. "Computer Animation". Computer Graphics 9 (Spring, 1975), pp. 92-101.

15. Csuri, Charles, Gomez, Julian, MacDougal, Paul, and Zeltzer, David. "A Tool Set for 3-D Computer Animation". Course Notes, Introduction to Computer Animation (July, 1984). ACM SIGGRAPH 84.

16. Donkin, John. Dime. Computer Graphics Research Group, The Ohio State University, Columbus, Ohio, 1984. undocumented program.

17. Foley, J. D. and Van Dam, A.. Fundamentals of Interactive Computer Graphics. Addison-Wesley, 1982.

18. Fortin, Denis, Lamy, Jean-Francois, and Thalman, Daniel. "A Multiple Track Animator System for Motion Synchronization". Proc. ACM SIGGRAPH/SIGART Workshop on Motion (April, 1983), pp. 180-186.

19. Girard, Michael and Maciejewski, A. A.. "Computational Modeling for the Computer Animation of Legged Creatures". Submitted for publication. Computer Graphics Research Group, The Ohio State University, 1985.

20. Gomez, Julian. "BBANI" in UNIX Programmer's Manual, Local Addendum. Computer Graphics Research Group, The Ohio State University, Columbus, Ohio,

21. Gomez, Julian. "Shapex" in UNIX Programmer's Manual, Local Addendum. Computer Graphics Research Group, The Ohio State University, Columbus, Ohio, October, 1983. A program that provides near real-time calculation and vector playback of shape interpolation between two objects.

22. Gomez, Julian. March, 1984. Personal communication.

23. Gomez, Julian. "Twixt: A 3D Animation System" in Proc. Eurographics '84. Copenhagen, September, 1984.

24. Gomez, Julian. February, 1985. Personal communication.

25. Gunther, Brian. "Using Parametric Space Curves in Computer Animation". Submitted for publication. Computer Graphics Research Group, The Ohio State University, January, 1985.

26. Gunther, Brian. March, 1985. Personal communication.

27. Ho, Shaun. "Poly_scan" in UNIX Programmer's Manual, Local Addendum. Computer Graphics Research Group, The Ohio State University, Columbus, Ohio, A scan-line, depth buffer rendering program.

28. Kingsbury, Douglas L.. Bubble check, Flightsim, Helix, Penbldr, Pen Combine. Computer Graphics Research Group, The Ohio State University, Columbus, Ohio, 1983. Undocumented special purpose programs.

29. Kochanek, Doris H. and Bartels, Richard H.. "Interpolating Splines with Local Tension, Continuity, and Bias Control". Computer Graphics 18, 3 (July, 1984). Proc. ACM SIGGRAPH 84.

30. Lasseter, John. December, 1984. Personal communication.

31. Longtin, Thomas J.. "Algorithm for Conforming Object Data to a Space Path". Unpublished paper. The Ohio State University, Autumn, 1984.

32. Lundin, Richard V.. "Motion Simulation". Nicograph 1984 Proceedings (November, 1984), pp. 2-10.

33. MacDougal, Paul. "Generation and Management of Object Description Hierarchies for the Simplification of Image Generation". Unpublished Ph.D. Dissertation. Computer Graphics Research Group, The Ohio State University, August, 1984.

34. Maciejewski, A. A. and Klein, C. A.. "SAM - Animation Software for Simulating Articulated Motion". Unpublished paper. Department of Electrical Engineering, The Ohio State University, 1984.

35. Meriam, J. L.. Mechanics. John Wiley and Sons, New York, 1959.

36. Mezei, L. and Zivian, A.. "ARTA, an Interactive Animation System". Proceedings, IFIP Congress (1971), pp. 429-434.

37. Muybridge, Eadweard. Animals in Motion. Dover, New York, 1957.

38. Newman, William A. and Sproull, Robert F.. Principles of Interactive Computer Graphics. McGraw-Hill Book Co., New York, 1979.

39. Palyka, Duane. Autumn, 1984. Personal Communication.

40. Parke, Frederick I.. "Parameterized Models for Facial Animation". IEEE Computer Graphics and Applications 2, 9 (November, 1982), pp. 61-68.

41. Platt, S. M. and Badler, Norman I.. "Animating Facial Expressions". Computer Graphics 15, 3 (August, 1981), pp. 245-252.

42. Porter, Thomas and Duff, Tom. "Compositing Digital Images". SIGGRAPH '84 Conference Proceedings (July, 1984). ACM SIGGRAPH '84.

43. Reeves, William T.. "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects". ACM Transactions on Graphics 2, 2 (April, 1983). Association for Computing Machinery.

44. Reynolds, Craig. "Computer Animation with Scripts and Actors". Computer Graphics 16, 3 (July, 1982), pp. 289-296. Proc. ACM SIGGRAPH 82.

45. Ribble, E. A.. "Synthesis of Human Skeletal Motion and the Design of a Special-Purpose Processor for Real-Time Animation of Human and Animal Figure Motion". M.S. Thesis. The Ohio State University, June, 1982.

46. Rogers, David F. and Adams, Alan J.. Mathematical Elements for Computer Graphics. McGraw-Hill Book Co., New York, 1976.

47. Sachter, Judy. "The Basic Concepts of Three-Dimensional Computer Graphics for Artists".

Unpublished Masters Thesis. Computer Graphics Research Group, The Ohio State University, June, 1984.

48. Schweppe, Marla. "Timing and Movement in 3-D Computer Animation". Unpublished M.A. Thesis. Computer Graphics Research Group, The Ohio State University, March, 1984.

49. Sharpe, Richard J. and Thorpe, Richard W.. "Numerical Method for Extracting an Arclength Parametrization from Parametric Curves". Computer Aided Design 12, 2 (March 1982), pp. 79-81.

50. Spegel, M.. "Programming of Mechanism Motion". Technical Report No. CRL-43. Division of Applied Science, New York University, New York, November, 1975.

51. Sutherland, Ivan, Sproull, Robert, and Schumacker, Robert. "A Characterization of Ten Hidden-Surface Algorithms". Tutorial: Computer Graphics (1982). IEEE Computer Society.

52. Thomas, Frank and Johnston, Ollie. Disney Animation: The Illusion of Life. Abbeville Press, New York, 1981.

53. Tripp, Edwin III. Tog (Tubular Object Generator). Computer Graphics Research Group, The Ohio State University, Columbus, Ohio, 1983. undocumented procedural object generator.

54. Unknown, Totally. "Tools for Motion Design". Course Notes, Seminar on Three Dimensional Computer Animation (July, 1982). ACM SIGGRAPH '82.

55. Van Baerle, Susan and Kingsbury, Doug. "Snoot and Muttly". SIGGRAPH Video Review 1984 15, (1984).

56. Van Baerle, Susan. "Computer-Generated, Three-Dimensional Character Animation". Unpublished Masters Thesis. Computer Graphics Research Group, The Ohio State University, March, 1985.

57. Wallace, Bruce A.. "Merging and Transformation of Raster Images for Character Animation". SIGGRAPH '81 Conference Proceedings (August, 1981). ACM SIGGRAPH '81.

58. Wein, Marceli, Barsky, Brian, Evans, Ken, and Fournier, Alain. Course Notes, Introduction to Computer Graphics July, 1983. ACM SIGGRAPH '83

59. Whitaker, H. and Halas, J.. Timing for Animation. Focal Press Limited, London, 1981.

60. Williams, Lance. "BBOP". Course Notes, Seminar on Three-Dimensional Computer Animation (July, 1982). ACM SIGGRAPH 82.

61. Williams, Lance. "Overview of 3D Animation". Course Notes, Seminar on Three-Dimensional Computer Animation (July, 1982). ACM SIGGRAPH 82.

62. Woo, Tony C.. "Progress in Shape Modeling". Computer 10, 12 (December, 1982), pp. 40-46.

63. Zeltzer, David. Representation and Control of Three Dimensional Computer Animated Figures. Computer Graphics Research Group, The Ohio State University, August, 1984. Unpublished Ph.D. Dissertation.

BIBLIOGRAPHY

Albers, Josef, Interaction of Color, Yale University Press, New Haven, Conn. (1975).

Arnheim, Rudolf, Art and Visual Perception: Psychology of the Creative Eye, University of California Press, Berkeley, Ca. (1974).

Badler, Norman I., "Design of a Human Movement Representation Incorporating Dynamics and Task Simulation," Course Notes, Seminar on Three-Dimensional Computer Animation ACM SIGGRAPH '82, Boston, MA. (July, 1982).

Barnhill, R. E. and Riesenfeld, R.F., Computer Aided Geometric Design, Academic Press, New York, (1976).

Blair, Preston, Animation, Walter Foster Art Books, Tustin, CA.

Blinn, James F., "Introduction to 3D Graphics", Z. Molnar, & V. Sorenson (Ed.), Course Notes, Seminar on The Artist/Designer and Computer Graphics, ACM SIGGRAPH '83, Detroit, Mich., (July, 1982).

Bradley, Gerald L., A Primer of Linear Algebra, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, (1975).

Dungan, William, Jr., "Computer Animation at Information International", ACM SIGGRAPH '82, Boston, MA. (July, 1982)

Forrest, A. R., "On Coons and Other Methods for the Representation of Curved Surfaces", Computer Graphics and Image Processing, 1, 1972, pp. 341-359.

Girard, Michael and Maciejewski, A. A., "Computational Modeling for the Computer Animation of Legged Figures", unpublished paper, The Ohio State University (January 1985).

Greenburg, Donald P., The Computer Image. ed. D. P. Greenburg, Addison-Wesley, (1982).

Muybridge, Eadweard, The Human Figure in Motion, Dover Publications, Inc., New York, (1955).

Parent, Richard E., "A System for Generating Three-Dimensional Data for Computer Graphics", unpublished Ph.D. dissertation, The Ohio State University, (December, 1977).

Smith, Alvy Ray, "Plants, Fractals, and Formal Languages", SIGGRAPH '84 Conference Proceedings, Minneapolis, MN. (July 1984).

Smith, Alvy Ray, "Special Effects for Star Trek II, The Genesis Demo, Instant Evolution with Computer Graphics", American Cinematographer, (October, 1982).

Stern, Garland, "Bbop - A Program for 3-Dimensional Animation", Tokyo, (December, 1983).

Stredney, Donald, "The Representation of Anatomical Structures Through Computer Animation for Scientific, Educational and Artistic Applications", unpublished Masters Thesis, The Ohio State University (March, 1982).

Sturman, David, "Interactive Keyframe Animation of 3-D Articulated Models", Proceedings of Graphics '84, Ottawa, Canada, (1984).

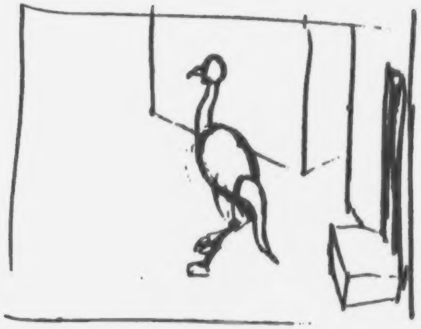
Wallace, Bruce, A., "Merging and Transformation of Raster Images for Cartoon Animation", SIGGRAPH '81 Conference Proceedings, Dallas, Texas, (August, 1981)

Williams, Lance, "BBOP", Course Notes, Seminar on Three-Dimensional Computer Animation, ACM SIGGRAPH '82, Boston, MA. (July, 1982).

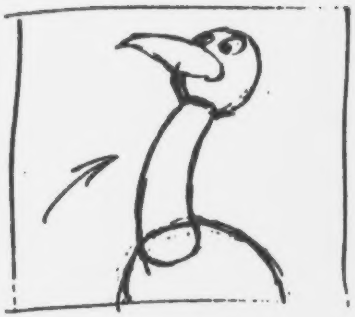
Zeltzer, David, "Three Dimensional Movement Simulation," Course Notes, Seminar on Three Dimensional Computer Animation, ACM SIGGRAPH '82, Boston Ma., (July, 1982).

APPENDIX A

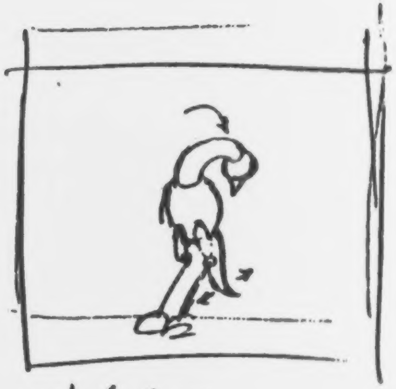
Storyboard Sketches



cut back to snoot
from nut's point-of-view
snoot's posture is perfect—
head, shoulders back
tail gracefully down



Nut pulls himself
together
head too far back
neck contrasts with
snoot's .
body back more than
snoot's — nut seems to
be on verge of falling
over backwards.



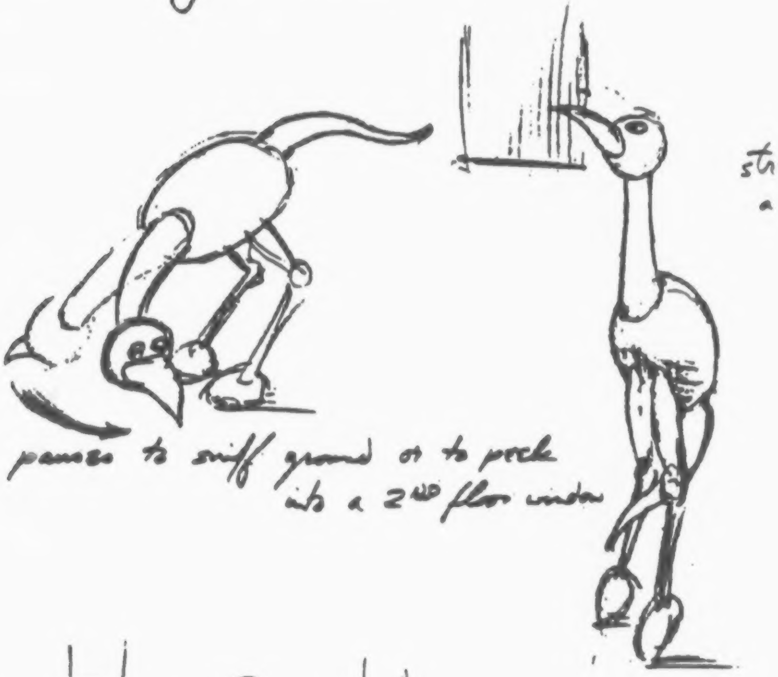
cut back —
nutty quickly checks
tail — which he wags to
emphasize
then turns around, pauses to
get himself mentally prepared
for this charade — then begins
to walk forward.

APPENDIX B

Business Sketches

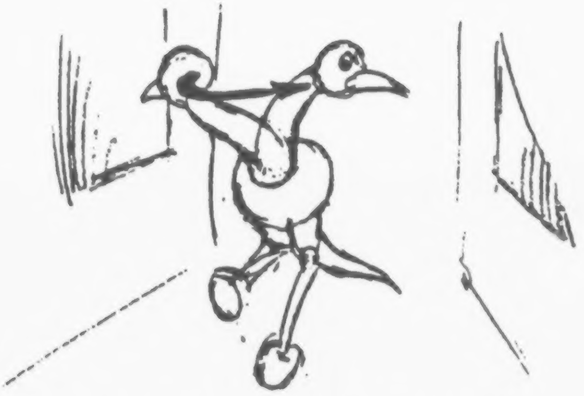
75
2

Mathy's entrance
very animated - neck & tail



stretch neck
a little

pauses to sniff ground or to peck
into a 2ND floor window

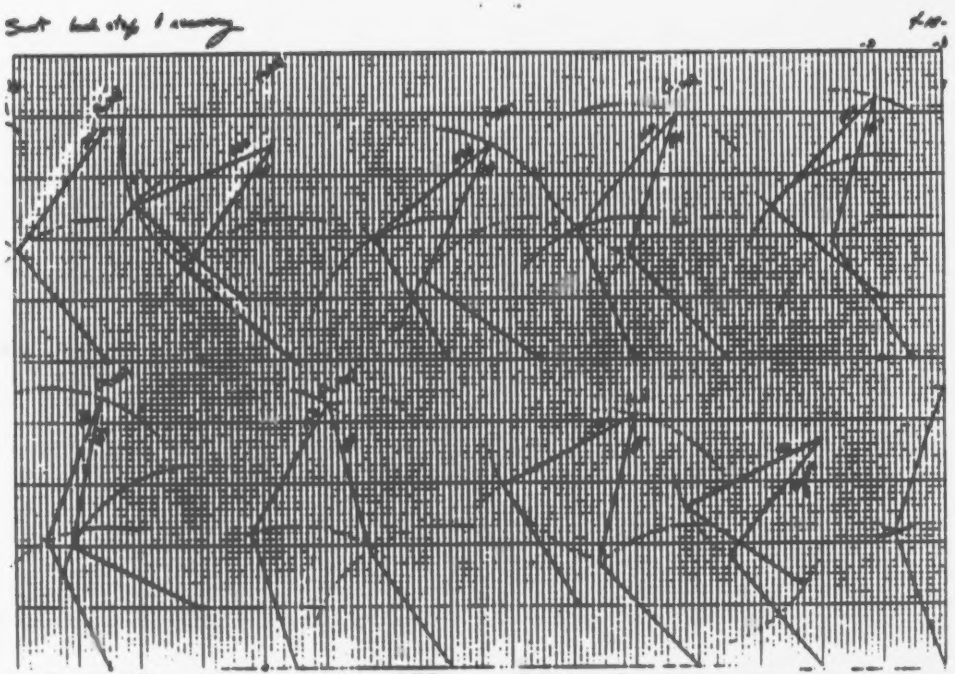


Eyes, head, neck & tail moving
constantly.

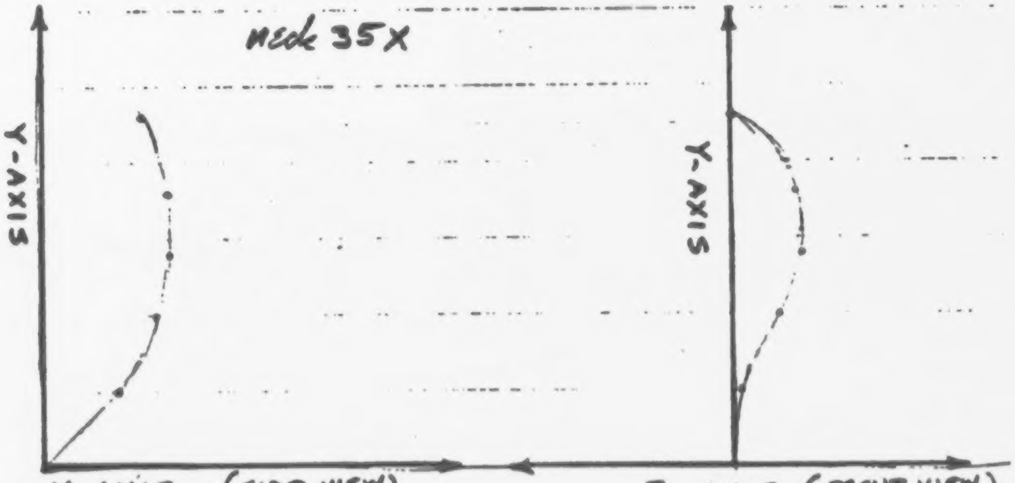
8 sec.

APPENDIX C

Plans for Data Control Points



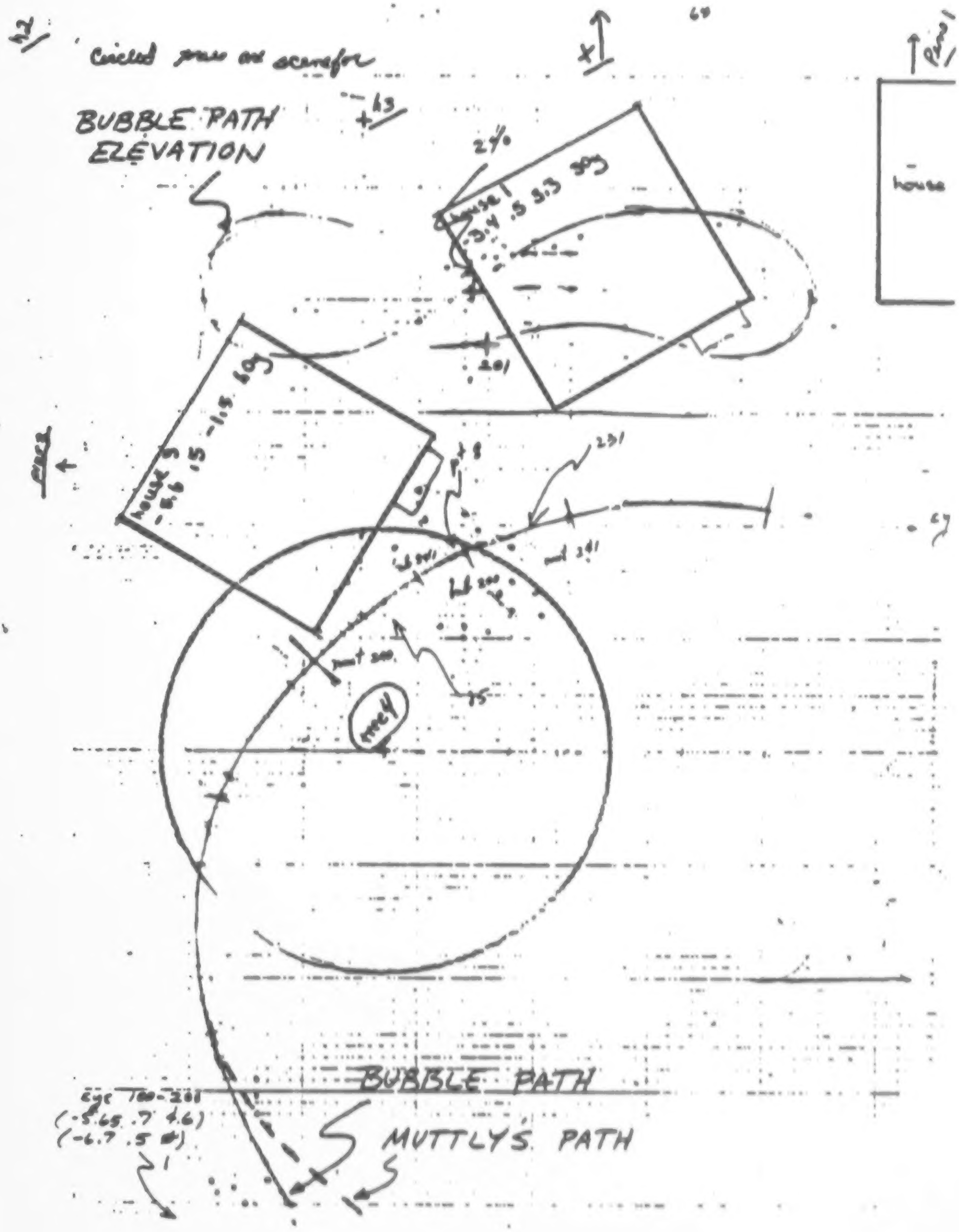
SNOOT'S BACKWARD STEP FROM BOTTOM RIGHT TO TOP LEFT


Neck 35X
Y-AXIS
Y-AXIS
X-AXIS (SIDE VIEW)
Z-AXIS (FRONT VIEW)

PLANS FOR NECK

APPENDIX D

Sample Movement Paths


Circled [illegible] are scenefor
BUBBLE PATH ELEVATION
X
house
house 1
-3.4 .5 3.3 30
house 3
-6.6 .5 -1.5 60
201
231
tree
75
BUBBLE PATH
MUTTLYS PATH
Eye 100-200
(-5.65 .7 4.6)
(-6.7 .5 4)
1